An Ensemble Architecture for Learning Complex Problem-Solving Techniques From Demonstration [*]

Xiaoqin (Shelley) Zhang[†], Sungwook Yoon[‡], Phillip DiBona[§], Darren Scott Appling[¶], Li Ding[‖], Janardhan Rao Doppa[**], Derek Green[††], Jinhong K. Guo[§], Ugur Kuter[‡‡], Geoff Levine[★], Reid L. MacTavish[⁎], Daniel McFarlane[§], James R Michaelis[‖], Hala Mostafa[⊛], Santiago Ontañón[⁎], Charles Parker[**], Jainarayan Radhakrishnan[⁎], Anton Rebguns[††], Bhavesh Shrestha[†], Zhexuan Song[◇], Ethan B. Trewhitt[¶], Huzaifa Zafar[⊛], Chongjie Zhang[⊛], Daniel Corkill[⊛], Gerald DeJong[★], Thomas G. Dietterich[**], Subbarao Kambhampati[‡], Victor Lesser[⊛], Deborah L. McGuinness[‖], Ashwin Ram[⁎], Diana Spears[††], Prasad Tadepalli[**], Elizabeth T. Whitaker[¶], Weng-Keen Wong[**], James A. Hendler[‖], Martin O. Hofmann[§], Kenneth Whitebread[§]

August 5, 2010

Abstract

We present a novel ensemble architecture for learning problem-solving techniques from a very small number of expert solutions and demonstrate its effectiveness in a complex real-world domain. The key feature of our "Generalized Integrated Learning Architecture" (GILA) is a set of integrated learning and reasoning (ILR) components, coordinated by a central meta-reasoning executive (MRE). The ILRs are *weakly coupled* in the sense that all coordination happens through the MRE. Each ILR learns independently from a small number of expert demonstrations of a complex task. During the performance, each ILR proposes partial solutions to subproblems posed by the MRE, which are then selected from and pieced together by the MRE to produce a complete solution. We describe the application of this novel learning and problem solving architecture to the domain of airspace management, where multiple requests for the use of airspaces need to be deconflicted, reconciled and managed automatically. Formal evaluations show that our system performs as well as or better than humans after learning from the same training data. Furthermore, GILA outperforms any individual ILR run in isolation, thus demonstrating the power of the ensemble architecture for learning and problem solving.

1 Introduction

We present a learning and problem-solving architecture that consists of an ensemble of subsystems that learn to solve problems from a very small number of expert solutions. Because human experts who can provide training solutions for complex tasks such as airspace management are rare and their time is expensive, our learning algorithms are required to be highly sample-efficient. Ensemble architectures such as bagging, boosting, and co-training have proved to be highly sample-efficient in classification learning (Breiman 1996; Freund and Schapire 1996; Blum and Mitchell 1998; Dietterich 2000b). Ensemble architectures have a long history in problem solving as well, starting with the classic Hearsay system to the more recent explosion of research in multi-agent systems (Erman et al. 1980; Weiss 2000). The best performing planning system in the learning track of the most recent planning competition is based on an ensemble architecture (Galvani et al. 2009). In this paper, we explore an ensemble learning approach for use in problem solving. Both learning and problem solving are exceptionally complicated in our domain due to the complexity of the task, the presence of multiple interacting subproblems, and the need for near-optimal solutions. Unlike in bagging and boosting, where a single learning algorithm is typically employed, our learning and problem-solving architecture has multiple heterogeneous learner-reasoners that learn from the same training data and use their learned knowledge to collectively

[*]Distribution Statement (Approved for Public Release, Distribution Unlimited). This material is based upon work supported by DARPA through a contract with Lockheed Martin (prime contract #FA8650-06-C-7605). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA, Lockheed Martin or the U.S. Government.

[†]University of Massachusetts at Dartmouth
[‡]Arizona State University
[§]Lockheed Martin Advanced Technology Laboratories
[¶]Georgia Tech Research Institute
[‖]Rensselaer Polytechnic Institute
[**]Oregon State University
[††]University of Wyoming
[‡‡]University of Maryland
[★]University of Illinois at Urbana
[⁎]Georgia Institute of Technology
[⊛]University of Massachusetts, Amherst
[◇]Fujitsu Laboratories of America

solve test problems. The heterogeneity of the learner-reasoners allows both learning and problem solving to be more effective because their biases are complementary and synergistic. This feature is especially attractive in complex domains where the designer is often not sure which heuristics and biases are the most appropriate, and different parts of the problem often yield to different representations and solution techniques. However, for the ensemble problem solving to be truly effective, the architecture must support a centralized coordination mechanism that can divide the learning and problem-solving tasks into multiple subtasks that can be independently solved, distribute them appropriately, and judiciously combine the results to produce a consistent overall solution to the whole problem.

In this paper, we present a learning and problem-solving architecture that consists of an ensemble of integrated learning and reasoning components (ILRs) coordinated by a central subsystem called "meta-reasoning executive" (MRE). Each ILR has its own specialized representation of problem-solving knowledge, a learning component and a reasoning component which are tightly integrated for optimal performance. We have considered the following three possible approaches to coordinate the ILRs through the MRE during both learning and performance.

1. *Independent learning and selected performance.* Each ILR independently learns from the same training data and performs on the test data. The MRE selects one out of all the competing solutions for each test problem.
2. *Independent learning and collaborative performance.* The learning is independent as before. However, in the performance phase, the ILRs share individual subproblem solutions and MRE selects, combines and modifies shared subproblem solutions to create a complete solution.
3. *Collaborative learning and performance.* Both learning and performance are collaborative, with multiple ILRs sharing their learned knowledge and their solutions to the test problems.

Roughly speaking, in the first approach, there is minimal collaboration only in the sense of a centralized control that distributes the training examples to all ILRs and selects the final solution among the different proposed solutions. In the second approach, learning is still separate, while there is stronger collaboration during the problem solving in the sense that ILRs solve individual subproblems, whose solutions are selected and composed by the central MRE. In the third approach, there is collaboration during both learning and problem solving.

The approach we describe in this project, namely, *independent learning with limited sharing and collaborative performance* is closest to the second approach. It is simpler than the third approach where learning is collaborative, and still allows the benefits of collaboration during performance by being able to exploit individual strengths of different ILRs. Since there is no requirement to share the learned knowledge, each ILR adopts an internal representation of knowledge and learning method that is most suitable to its own performance. Limited sharing of learned knowledge does happen in the GILA system, though it is not required in this architecture. There are two ILRs sharing their learned constraint knowledge. This is possible because they use the same internal representation format for constraint knowledge; therefore, no extra conversion effort is needed (see Section 6.3.1 for more details).

The ILRs use shared and ILR-specific knowledge in parallel to expand their private databases. The MRE coordinates and controls the learning and the performance process. It directs a collaborative search process, where each search node represents a problem-solving state and the operators are subproblem solutions proposed by ILRs. MRE uses the learned knowledge provided by ILRs to decide the following: (1) which subproblem to work on next, (2) which subproblem solution (search node) to select for exploration (expand) next, (3) when to choose an alternative for a previous subproblem that has not been explored yet, and (4) when to stop the search process and present the final solution. In particular, our Generalized Integrated Learning Architecture (GILA) offers the following features:

- Each ILR learns from the same training data independently of the other ILRs, and produces a suitable hypothesis (solution) in its own language.
- A blackboard architecture (Erman et al. 1980) is adopted to enable communication among the ILRs and the MRE.
- During the performance phase, the MRE directs the problem-solving process by subdividing the overall problem into subproblems and posting them on a centralized blackboard structure.
- Using the prioritization knowledge learned by one of the ILRs, the MRE directs the ILRs to work on one subproblem at a time. Subproblems are solved independently by each ILR, and the solutions are posted on the blackboard.
- The MRE conducts a search process, using the subproblem solutions as operators, in order to find a path leading to a conflict-free goal state. The path combines appropriate subproblem solutions to create a solution to the overall problem.

There are several advantages of this architecture. They are listed below.

- *Sample Efficiency.* This architecture could result in rapid learning, since each example may be used by different learners to learn from different small hypothesis spaces. This is especially important when the training data is sparse and/or expensive.
- *Semi-supervised learning.* The learned hypotheses of our ILRs are diverse even though they are learned from the same set of training examples. Their diversity is due to multiple learning algorithms. Therefore, we can leverage unlabeled examples in a

co-training framework (Blum and Mitchell 1998). Multiple learned hypotheses could improve the solution quality, if the MRE is able to select the best from the proposed subproblem solutions and compose them.

- *Modularity and Extensibility.* Each ILR has its own learning and reasoning algorithm, it can use specialized internal representations that it can efficiently manipulate. The modularity of different ILRs make it easier to integrate new ILRs into the system in a plug-and-play manner, since they are not required to share the same internal representations.

We applied this learning and problem-solving architecture to an airspace management problem that challenges even the best human experts because it is complex and knowledge-intensive. Not only do experts need to keep track of myriad details of different kinds of aircraft and their limitations and requirements, but they also need to find a safe, mission-sensitive and cost-effective global schedule of flights for the day. An expert system approach to airspace management requires painstaking knowledge engineering to build the system, as well as a team of human experts to maintain it when changes occur to the fleet, possible missions, safety protocols and costs of schedule changes. For example, flights need to be rerouted when there are forest fires occurring on their original paths. Such events require knowledge re-engineering. In contrast, our approach based on learning from an expert's demonstration is more attractive, especially if it only needs a very small number of training examples, which are more easily provided by the expert.

We developed a system called *Generalized Integrated Learning Architecture* (GILA) that is evaluated in the domain of airspace management (Section 2). GILA implements this ensemble learning architecture (Section 3) for complex problem solving. In GILA system, various components communicate using a common ontology language (Section 4). Components in GILA include the MRE (Section 5) and four different ILRs with diverse biases: the symbolic planner learner-reasoner (SPLR) (Section 6.1), the decision-theoretic learner-reasoner (DTLR) (Section 6.2), the case-based learner-reasoner (CBLR) (Section 6.3), and the 4D-deconfliction and constraint learner-reasoner (4DCLR) (Section 6.4). In rigorously evaluated comparisons (Section 7), GILA was able to outperform human novices who were provided with the same background knowledge and the same training examples as GILA, and GILA used much less time than the human novices. Our results show that the quality of the solutions of the overall system is better than that of any individual ILR. Related work is presented in Section 8. Our work demonstrates that the ensemble learning and problem-solving architecture as instantiated by GILA is an effective approach to learning and managing complex problem solving in domains such as airspace management. We summarize The lessons learned from this work and how GILA can be transferred to other problem domains are discussed in Section 9.

2 Domain Background

The domain of application for the GILA project is airspace management in an Air Operations Center (AOC). Airspace management is the process of making changes to requested airspaces so that they do not overlap with other requested airspaces or previously approved airspaces. The problem that GILA tackles is the following. Given a set of Airspace Control Means Requests (ACMReqs), each representing an airspace requested by a pilot as part of a given military mission, identify undesirable conflicts between airspace uses and suggest changes in latitude, longitude, time or altitude that will eliminate them. An Airspace Control Order (ACO) is used to represent the entire collection of airspaces to be used during a given 24-hour period. Each airspace is defined by a polygon described by latitude and longitude points, an altitude range and a time interval during which the air vehicle will be allowed to occupy the airspace.

For example, airspace ACM-J-01 is described as:
ACM-J-01
Usage: AEW (Airborne Early Warning Area)
Shape: orbit
Min Altitude: 25500.0
Max Altitude: 35500.0
Start Time: 2007-06-21T00:00:00.0
End Time: 2007-06-21T23:59:00.0
Radius: 10.0
Coordinates: [Point 0: 37.340,-116.984; Point 1: 36.860,-116.580]

The process of deconfliction assures that any two vehicles' airspaces do not overlap or conflict. Figure 1 shows an example of an original ACO and Figure 2 shows the modified ACO after the deconfliction actions of the expert, who is also called the *Subject Matter Expert (SME)*. Each polygon, either in yellow or light blue color, represents an airspace request - an Airspace Control Mean (ACM). Those in yellow are existing airspaces that cannot be modified. The black line is the Forward Edge of the Battle Area (FEBA) line, which defines the foremost limits of a series of areas in which ground combat units are deployed. There are 14 conflicts in the original ACO shown in red or pink; the one in red is currently being selected by the user. The expert removes all these conflicts through the modifications described in Table 1. Notice that each ACM is defined in four dimensions (three spatial

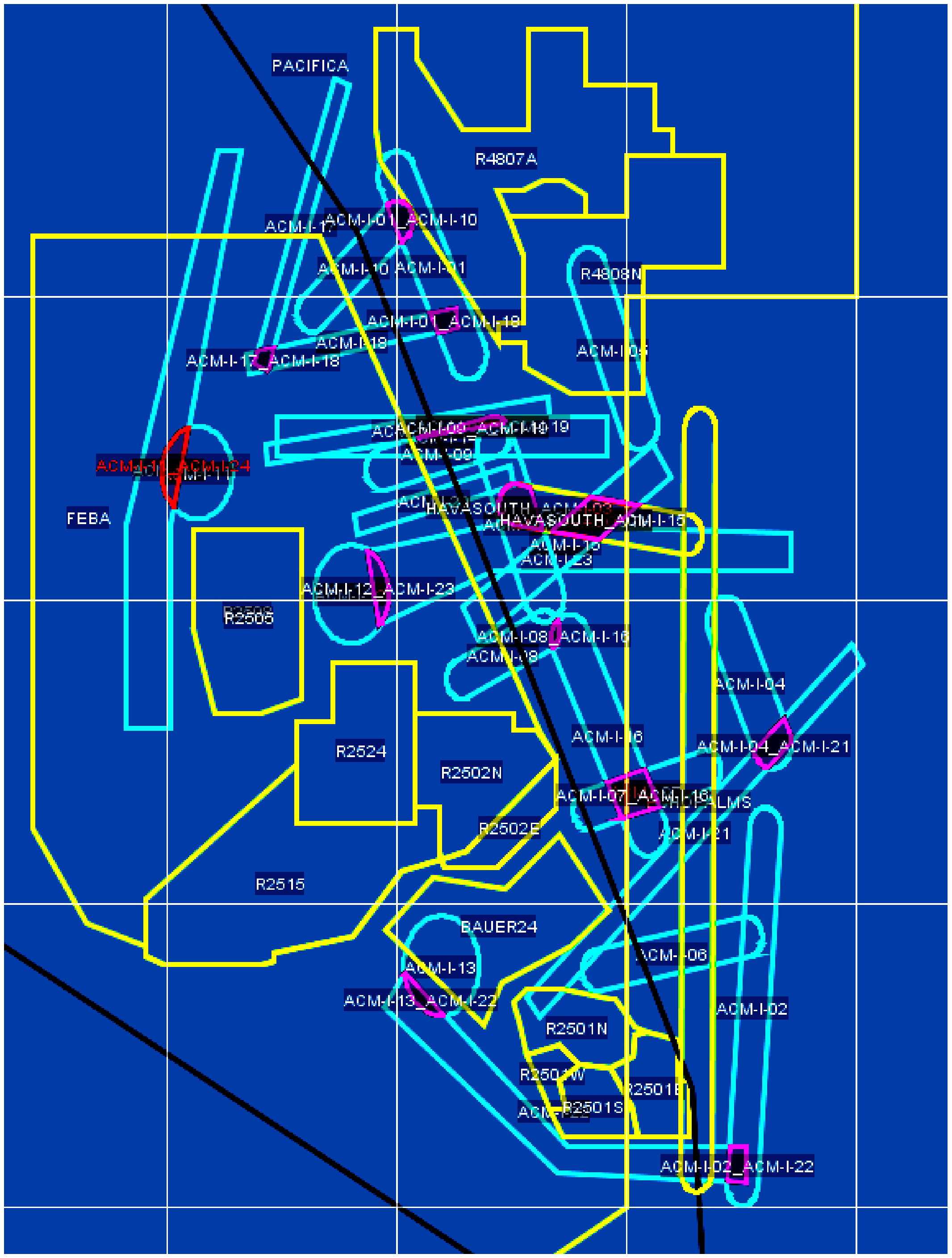


Figure 1: Example of an ACO in Scenario E consisting of 24 ACMs and 14 conflicts. Yellow: existing airspaces that cannot be modified.

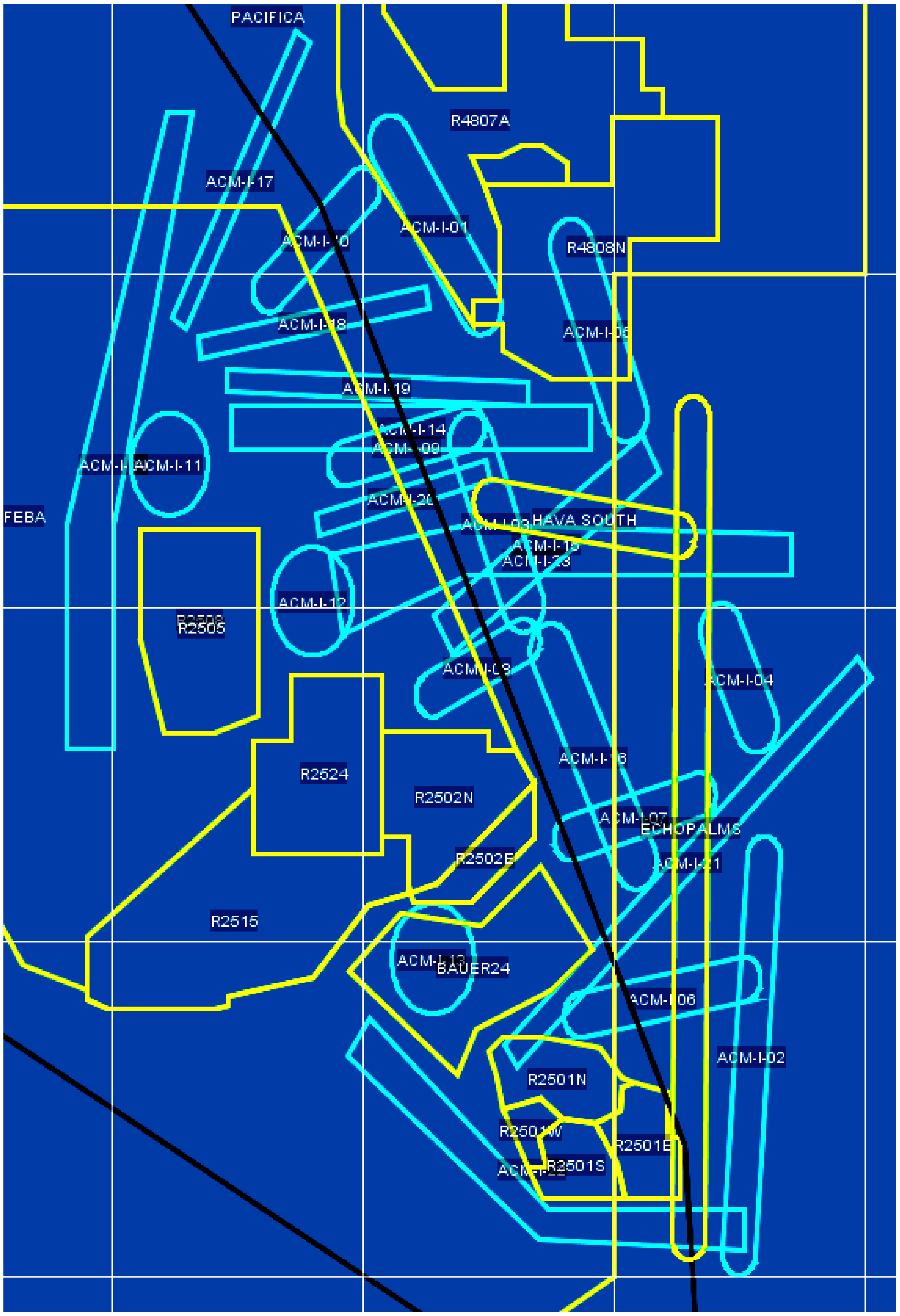


Figure 2: ACO in Scenario E after the expert's deconfliction.

Table 1: The deconfliction solution trace for Scenario E from the Subject Matter Expert (SME).

Conflict priority	Conflict	Expert's Solution
1	ACM-I-17 (SSMS) ACM-I-18 (SSMS)	Move 17 position from 37.71/-117.24, 36.77/-117.59 to 37.71/-117.24, 36.85/-117.73
2	ACM-I-1 (AEW) ACM-I-18 (SSMS)	Move 1 position from 37.40/-117.00, 36.81/-116.70 to 37.39/-116.89, 36.9/-116.53.
3	ACM-I-3 (ABC) ACM-I-19 (SOF)	Move 19 position from 36.64/-116.34, -117.57/-36.48 to 36.64/-116.34, 36.68/-117.54
4	ACM-I-9 (CAP) ACM-I-19 (SOF)	Already Changed
5	ACM-I-13 (CASHA) ACM-I-22 (UAV)	Move 13 position from 34.80/-116.81, to 34.83/-116.76
6	ACM-I-2 (AEW) ACM-I-22 (UAV)	Change 22 MaxAltitude from 22500.0 to 19500.0.
7	HavaSouth (AAR) ACM-I-15 (COZ)	Change time of 15 from 1100-2359 to 1200-2359
8	HavaSouth (AAR) ACM-I-3 (ABC)	Change time of 3 from 1100-2359 to 1200-2359
9	ACM-I-8 (CAP) ACM-I-16 (RECCE)	Change 8 time from 1630-2100 to 1630-1959
10	ACM-I-7 (CAP) ACM-I-16 (RECCE)	Change 16 alt from 19500-28500 to 19500-27500
11	ACM-I-4 (AAR) ACM-I-21 (SSMS)	Move 4 position from 35.53/-115.37, 35.93/-115.57 to 35.65/-115.43, 35.93/-115.57
12	ACM-I-11 (CASHA) ACM-I-24 (UAV)	Move 11 position from 36.42/-117.87 to 36.43/-117.77
13	ACM-I-12 (CASHA) ACM-I-23 (UAV)	Change 23 Altitude from 1500-18000 to 1500-17000
14	ACM-I-1 (AEW) ACM-I-10 (CAP)	Already resolved

and one temporal), and therefore the spatial overlap shown in the 2D graph may or may not be a conflict. Table 1 shows an expert solution trace that GILA can use for training. According to this expert trace, in order to resolve a conflict that involves two ACMs, the expert first chooses one ACM and then decides whether to move its position, change its altitude or change its time.

To solve the airspace management problem, GILA must decide in what order to address the conflicts during the problem-solving process and, for each conflict, which airspace to move and how to move the airspace to resolve the conflict and minimize the impact on the mission. The main problem in airspace deconfliction is that there are typically infinitely many ways to successfully deconflict. To resolve a particular conflict, some changes are better than others according to the expert's internal domain knowledge; however, such knowledge is not revealed to GILA, and the only input from which GILA may learn is one expert trace such as the one shown in Table 1. The goal of the system is to find deconflicted solutions that are qualitatively *similar to* those found by human experts. Because it is hard to specify what this means, we take the approach of learning from the expert's demonstrations, and evaluate GILA's results by comparing them to human solutions.

3 Ensemble Learning Architecture for Complex Problem Solving

In this section, we will give a schematic view of our *Ensemble Architecture*, briefly state the reasons behind our choice of this architecture and explain its usefulness in a variety of different settings.

3.1 Problem Statement

Given a small set of training demonstrations $\{\langle \mathcal{P}_i, \mathcal{S}_i \rangle\}_{i=1}^{m}$ of task $\mathcal{T}$ to solve a complex problem, we want to learn the general problem-solving skills for the task $\mathcal{T}$. For example, in our airspace management problem which is described above, each problem $\mathcal{P}_i$ encodes a set of conflicts introduced by the Airspace Control Means Requests (ACMReqs) with the existing Airspace Control Order (ACO) and the correspoding solution $\mathcal{S}_i$ gives a sequence of deconfliction steps that needs to be executed to achieve a conflict-free Airspace Control Order (ACO).

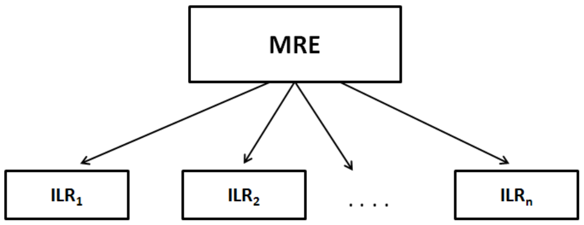


Figure 3: Ensemble Learning Architecture: ILR stands for Independent Learner-Reasoner and MRE stands for Meta Reasoning Executive.

3.2 Ensemble Architecture

Most of the traditional ensemble learning algorithms for classification like bagging or boosting, use a single hypothesis space and a single learning method. We use multiple hypotheses spaces and multiple learning methods in our architecture corresponding to each Independent Learner-Reasoner(ILR), and a Meta Reasoning Executive(MRE) that combines the decisions from the ILRs. Figure 3 shows our ensemble architecture. We view solving each problem instance of the given task $\mathcal{T}$ as a state-space search problem. The start state S consists of a set of subproblems $sp_1, sp_2, \ldots sp_k$. At each step, MRE chooses a subproblem sp_i and then give that chosen subproblem to each ILR for solving. ILRs return their solutions for the given subproblem, and MRE then picks the best solution. This process repeats until all the subproblems are solved or in other words we reach a goal state. For example, in our airspace management problem, each subproblem sp_i is a conflict involving airspaces.

Connections to Search-based Structured Prediction: Our approach can be viewed as a general version of *Search-based Structured Prediction*. The general framework of search-based structured prediction (Daumé III and Marcu 2005; Daumé III, Langford, and Marcu 2009) views the problem of labeling a given structured input x by a structured output y as searching through an exponential set of candidate outputs. LaSo (Learning as Search optimization) was the first work in this paradigm. *LaSo* tries to learn a heuristic function that guides the search to reach the desired output y quickly based on all the training examples. Xu et al. extended this framework to learn beam search heuristics for planning problems (Xu, Fern, and Yoon 2007). In the case of greedy search (Daumé III, Langford, and Marcu 2009), the problem of predicting the correct output y for a given input x can be seen as making a sequence of smaller predictions $y_1, y_2, \ldots, y_T$ with each prediction y_i depending on the previous predictions. It reduces the structured prediction problem to learning a multi-class classifier h that predicts the correct output y_t at time t based on the input x and partial output $y_1, y_2, \ldots, y_{t-1}$. In our case, each of these smaller predictions y_i corresponds to solutions of the subproblems sp_i, which can be more complex (structured outputs) than a simple classification decision.

Independent Learner-Reasoner(ILR): We have multiple ILRs as shown in Figure 3. Each ILR learns how to solve subproblems sp_i from the given set of training demonstrations $\{\langle \mathcal{P}_i, \mathcal{S}_i \rangle\}_{i=1}^{n}$ for task $\mathcal{T}$. We have diverse ILRs, each of which is using a different hypothesis space and/or learning method.

Meta Reasoning Executive(MRE): MRE is the decision maker in our ensemble architecture. It makes decisions such as which subproblem sp_i to pick next (search-space ordering) and which subproblem solution to pick among all the candidates provided by ILRs (evaluation).

3.3 Demonstration Learning - Individual ILR Learning

We are provided with a set of training examples (demonstrations) $\{\langle \mathcal{P}_i, \mathcal{S}_i \rangle\}_{i=1}^{m}$ of task $\mathcal{T}$ and the corresponding training examples $\{\langle \mathcal{P}_i, \mathcal{R}_i \rangle\}_{i=1}^{m}$ of ranking the subproblems when performing task $\mathcal{T}$. Learning inside our ensemble architecture happens as follows. First, we learn a ranking function $\mathcal{R}$ using a rank-learning algorithm. Then we learn the hypothesis h_{ILR_i} for each ILR i from the given training examples which we call *Individual ILR Learning*. We will describe the learning methodology of each ILR in Section 6. Note that, ILRs are diverse as they use different hypothesis space and/or learning method.

3.4 Ensemble Solving - Collaborative Performance

We describe how to solve a new problem $\mathcal{P}$ for task $\mathcal{T}$ (performance phase) in Algorithm 1. We initialize the start state s to the set of subproblems $sp_1, sp_2, \ldots sp_k$. We pick the highest ranked subproblem sp_{chosen} based on our learned ranking function $\mathcal{R}$. MRE gives the chosen subproblem sp_{chosen} to all the ILRs and each ILR i returns its solution sol_{ILR_i}. MRE picks the best subproblem solution sol_{best} by evaluating the states resulting from applying each of these subproblem solutions to the current state, based on a domain specific evaluation function $\mathcal{E}$ that was created using the learned knowledge. We update the state by applying the solution sol_{best} to the current state. We repeat this until we reach the goal state i.e., all our subproblems are solved or a maximum cut-off time is reached. If we reach one such goal state, then we return the sequence of subproblem solutions applied from start state to goal state, otherwise we return *no solution found*.

Algorithm 1 ENSEMBLE SOLVE

```
Input: problem instance $\mathcal{P}$ of task $\mathcal{T}$
learned hypothesis of each ILR $h_{ILR_1}, h_{ILR_2}, \ldots h_{ILR_n}$
ranking function $\mathcal{R}$ to rank subproblems
evaluation function $\mathcal{E}$ to evaluate the subproblem solutions provided by ILRs
Output: solution of the given problem $sol$

1: Initialize the start state $s = sp_1, sp_2, \ldots sp_k$
2: repeat
3:     $sp_{chosen}$ = highest ranked subproblem in the current state based on $\mathcal{R}$
4:     for each ILR $i = 1$ to $n$ do
5:         Solve subproblem: $sol_{ILR_i} = SolveH_i(sp_{chosen})$
6:     end for
7:     Evaluate the Solutions : $sol_{best}$ = best solution based on the evaluation function $\mathcal{E}$
8:     Update State : $s'$ = new state obtained from applying $sol_{best}$ to the current state
9: until reaching goal state or maximum cut-off time is reached
10: if $s'$ is goal state then
11:     $sol$ = sequence of subproblem solutions applied from start state to goal state
12: else
13:     $sol = null$
14: end if
15: return solution of the given problem $sol$
```

3.5 Practice Learning

Algorithm 2 PRACTICE LEARNING

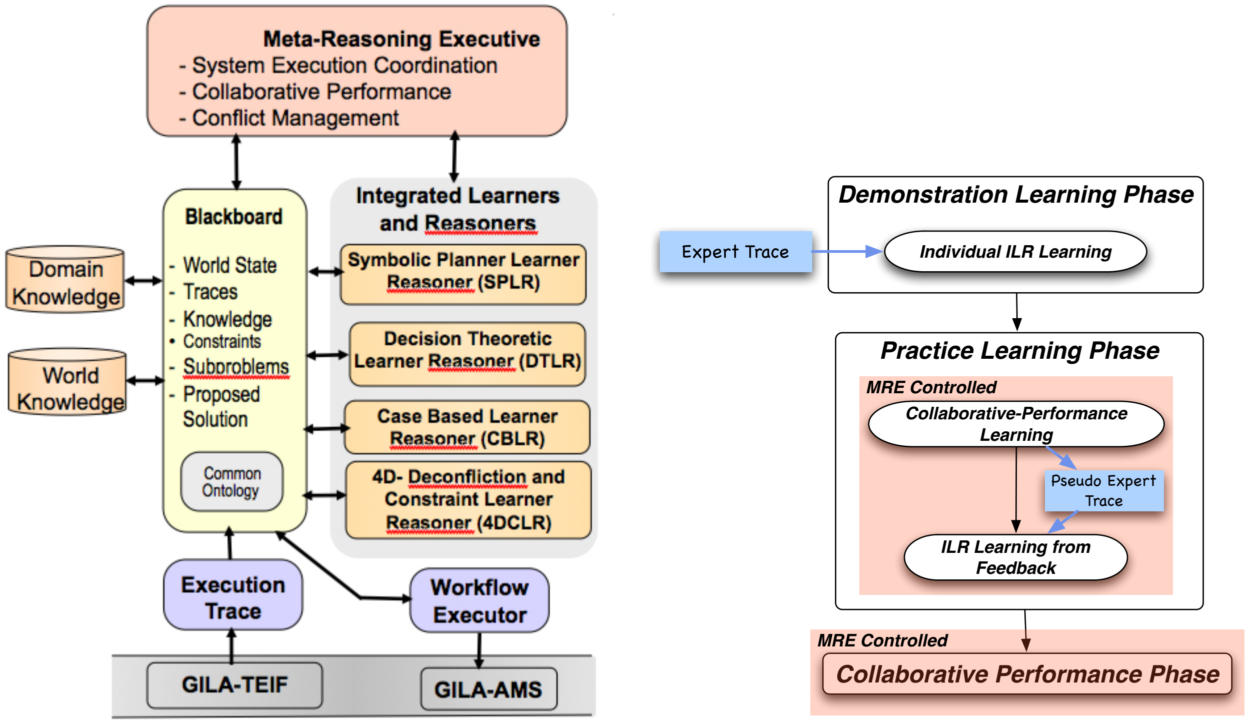
```
Input: $\mathcal{L}_p = \{\langle \mathcal{P}_i, \mathcal{S}_i \rangle\}_{i=1}^m$ the set of training demonstrations of task $\mathcal{T}$
$\mathcal{L}_r = \{\langle \mathcal{P}_i, \mathcal{R}_i \rangle\}_{i=1}^m$ the set of training examples for learning to rank subproblems
$\mathcal{U}$ = set of unsolved problem instances for task $\mathcal{T}$
Output: the learned hypothesis of each ILR $h_{ILR_1}, h_{ILR_2}, \ldots h_{ILR_n}$ and ranking function $\mathcal{R}$ to rank subproblems

1: Learn hypotheses $h_{ILR_1}, h_{ILR_2}, \ldots, h_{ILR_n}$ from solved training examples
2: Learn Ranking function $\mathcal{R}$ = Learn-Rank($\{\langle \mathcal{P}_i, \mathcal{R}_i \rangle\}_{i=1}^m$)
3: repeat
4:     $\mathcal{L}_{new} = \mathcal{L}_p$
5:     for each problem $\mathcal{P} \in \mathcal{U}$ do
6:         $\mathcal{S}$ = Ensemble-Solve($\mathcal{P}$, $h_{ILR_1}, h_{ILR_2}, \ldots, h_{ILR_n}$, $\mathcal{R}$, $\mathcal{E}$)
7:         $\mathcal{L}_{new} = \mathcal{L}_{new} \bigcup \langle \mathcal{P}, \mathcal{S} \rangle$
8:     end for
9:     Re-learn $h_{ILR_1}, h_{ILR_2}, \ldots, h_{ILR_n}$ from new examples $\mathcal{L}_{new}$
10: until convergence or maximum co-training iterations
11: return the learned hypothesis of each ILR $h_{ILR_1}, h_{ILR_2}, \ldots h_{ILR_n}$ and ranking function $\mathcal{R}$
```

In practice learning, we want to learn from a small set of training examples $\langle \mathcal{L}_p, \mathcal{L}_r \rangle$ and a large amount of unsolved problems $\mathcal{U}$. Our ideas are inspired by the iterative co-training algorithm (Blum and Mitchell 1998). The key idea in co-training is to take two diverse learners and make them learn from each other using the unlabeled data. In particular, co-training trains two learners h_1 and h_2 separately on two sufficient and redundant views ϕ_1 and ϕ_2, each of which is sufficient enough to learn a strong learner and conditionally independent of the other given the class label. Each learner will label some unlabeled data to augment the training set of the other learner and then, both learners are re-trained on this new training set and this process is repeated for several rounds. The difference or diversity between the two learners helps in teaching each other. As the co-training process proceeds, the two learners will become more and more similar, and the difference between the two learners becomes smaller. More recently, (Wang and Zhou 2007) proved a result which shows why co-training without redundant views can work. They show that as long as learners are diverse co-training will improve the performance of learners.

(Doppa and Tadepalli 2010) extended these ideas to search-based structured prediction for three learners and also n learners in the democratic learning framework (Goldman and Zhou 2004). In our case, diversity of our learners (ILRs) is due to multiple learning algorithms. First, we learn in a supervised framework with training demonstrations ($\mathcal{L}_p$, $\mathcal{L}_r$). Next, we solve each each problem $\mathcal{P} \in \mathcal{U}$ using Algorithm 1 and add it along with its solution to the training set. We re-learn using the new training set and repeat this until convergence or maximum co-training iterations.

3.6 Architecture Implementation and System Process



Meta-Reasoning Executive
- System Execution Coordination
- Collaborative Performance
- Conflict Management
Blackboard
- World State
- Traces
- Knowledge
• Constraints
- Subproblems
- Proposed Solution
Common Ontology
Domain Knowledge
World Knowledge
Integrated Learners and Reasoners
Symbolic Planner Learner Reasoner (SPLR)
Decision Theoretic Learner Reasoner (DTLR)
Case Based Learner Reasoner (CBLR)
4D- Deconfliction and Constraint Learner Reasoner (4DCLR)
Execution Trace
Workflow Executor
GILA-TEIF
GILA-AMS
Demonstration Learning Phase
Expert Trace
Individual ILR Learning
Practice Learning Phase
MRE Controlled
Collaborative-Performance Learning
Pseudo Expert Trace
ILR Learning from Feedback
MRE Controlled
Collaborative Performance Phase

(a) GILA's system architecture (b) GILA's system process

Figure 4: GILA's system architecture and process overview

GILA system implements the ensemble learning architecture as shown in Figure 4(a). We selected four different learner-reasoners (ILRs) for the GILA system ensemble. The first ILR is a *symbolic planner learner-reasoner (SPLR)*, which learns the expert's reactive strategy (what to do) and represents this knowledge as a set of decision rules. It also learns detailed tactics (how to do it) represented as value functions. This hierarchical learning closely resembles the reasoning process that a human expert uses when solving the problem. The second ILR is a *decision-theoretic learner-reasoner (DTLR)*, which learns a close approximation to the expert's cost function used in choosing among alternative solutions. This cost function is useful for GILA decision-making, assuming that the expert's solution optimizes the cost function subject to certain constraints. The DTLR is especially suitable for the types of problems that GILA is designed to solve. These problems generate large search spaces because each possible action has several numerical parameters that must be considered. The DTLR's learned knowledge is used by the MRE to evaluate the subproblem solution candidates provided by each ILR. The third ILR is a *case-based learner-reasoner (CBLR)*. It learns and stores a feature-based case database. The CBLR is good at learning aspects of the expert's problem solving that are not necessarily explicitly represented, storing the solutions and cases and applying this knowledge to solve similar problems. The last ILR is a *4D-deconfliction and constraint learner-reasoner (4DCLR)*, which learns and applies planning knowledge in the form of safety constraints. Such constraints are crucial in the airspace management domain that is the focus of this work. The 4DCLR is also used for internal simulation to generate the resulting state, in particular, to find the remaining conflicts after applying a subproblem solution. Each ILR has unique advantages, and the ensemble works together under the direction of the MRE to achieve the system's goals. No single ILR can perform as well as the whole GILA system, as is shown by the experimental results presented in Section 7.3.1.

The four ILR components and the MRE communicate through a blackboard using a common ontology. The blackboard holds the current world state, the expert's execution trace, some learned knowledge such as constraints, subproblems that need to be solved and the proposed partial solutions from ILRs. The expert performs on the GILA-TEIF (Training and Evaluation Interface) and the generated execution trace is loaded to the blackboard as the system input. The final solution produced by the system is executed by the workflow executor on the GILA-AMS (Air Management System). We will discuss the ontology, ILRs and MRE in detail in later sections.

As shown in Figure 4(b), the system process is divided into three phases: the *demonstration learning*, *practice learning* and *collaborative performance* phases. During the demonstration learning phase, a complete, machine-parsable trace of the expert's interactions with a set of application services is captured and made available to the ILRs via the blackboard. Each ILR uses shared world, domain and ILR-specific knowledge to expand its private models, both in parallel during individual learning and in collaboration. During the practice learning phase, GILA is given a practice problem (i.e., a set of airspaces with conflicts) and a goal state (with no remaining conflicts) but not told how this goal state is achieved (via actual modifications to those airspaces). The MRE then directs all ILRs to collaboratively solve this practice problem and generate a solution that is referred to as a "pseudo expert trace." ILRs can learn from this pseudo expert trace (assuming it is correct), thus indirectly sharing their learned knowledge through practice. In the collaborative performance phase, GILA solves a problem based on the knowledge it has already learned. The MRE decomposes the problem into a set of subproblems (where each subproblem is a conflict to resolve). These subproblems are then ordered using the CBLR's prioritization case library, which was learned by extracting an expert's order in solving problems. The ILRs are then requested to solve those subproblems, one at a time, according to the ordering. The MRE evaluates the partial solutions to the subproblems produced by different ILRs, and composes the best partial solutions from the ILRs into a complete final solution. The MRE's evaluation criteria include the knowledge learned by ILRs, such as the safety constraint violation information from 4DCLR and an execution cost factor that measures how close a partial solution is to the expert's demonstration from DTLR; details will be presented in Section 5. Using the learned knowledge, the MRE selects the best subproblem solution to explore first – until a path to the goal (conflict-free) state is found. The MRE composes the "best" partial solutions, subject to these inputs from the 4DCLR and DTLR, to yield an overall solution to the original problem.

4 Ontology and Explanations

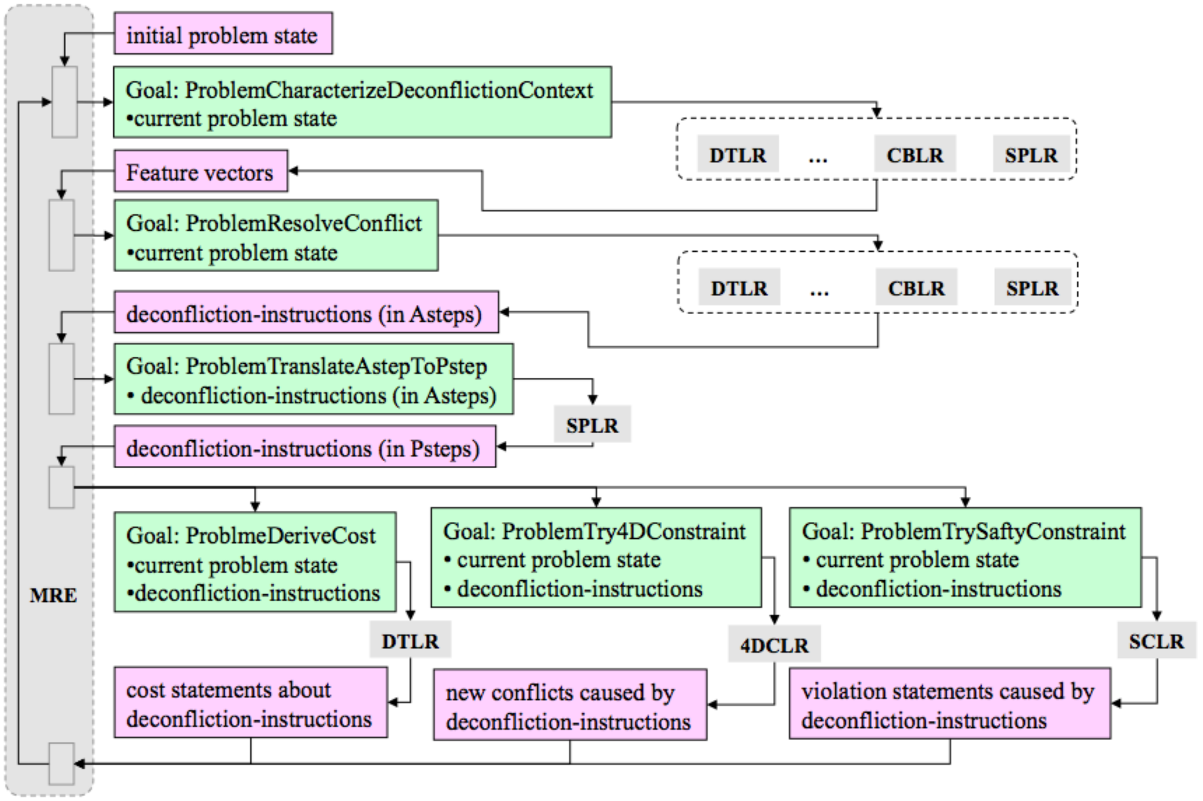
Background ontologies are used to aid GILA's learning, problem solving and explanation processes. To support the airspace deconfliction task described in Section 2, the GILA Inter-component Language (GIL) was designed as a family of GILA ontologies. The GILA ontologies are organized into two inter-dependent groups: domain knowledge ontologies and world knowledge ontologies. Domain knowledge ontologies are designed to support the learning and solving of the ACO deconfliction task in GILA, and include: (1) the GIL-ACO domain ontology (aka. *gilaco*), which helps describe ACO deconfliction specific problems and solutions, (2) the constraint ontology for describing spatiotemporal safety constraints, (3) the Abstract/Partial-plan/Sensing steps ontology for describing different types of deconfliction planning steps and (4) the Airspace Control Order (ACO) ontology for describing ACO domain concepts. World knowledge ontologies that are included for capturing domain-independent information are: (1) the Proof Markup Language (PML) ontologies for capturing provenance attributes and relations over GILA activities, (2) a general-purpose data structure ontology for describing data structures like lists, (3) a spatiotemporal ontology (aka. *GKST*) for describing temporal/spatial entities and (4) the GIL-CORE ontology (aka. *gilcore*) for describing generic problem-solving concepts.

GIL facilitates communication and collaboration between the MRE and the ILRs, despite their different implementation strategies. Instead of using a peer-to-peer communication model, the ILRs are driven by knowledge shared on a centralized blackboard. The ILRs and the MRE use the blackboard to obtain the current system state and potential problems, compute proposals for the next action and then publish their results. In addition, multiple ILRs can join the problem-solving process without altering the results of other ILRs.

An example workflow enabled by the above model is illustrated in Figure 5. The "Goal" rectangles refer to new system state updates that occur when a *problem* is posted on the blackboard, and the other rectangles refer to system state updates that occur when a *solution* is posted on the blackboard. The GILA components (e.g., ILRs and the MRE) are identified by the round-angle rectangles. Each GILA ILR accepts a selection of system updates from the blackboard (e.g., a new problem or solution), and then takes actions according to its capabilities. For example, the DTLR can handle the following types of problems defined in the gilaco ontology:

- *gilaco:ProblemCharacterizeDeconflictionContext* - ask an ILR to characterize the current problem state using a feature vector.
- *gilaco:ProblemResolveConflict* - ask an ILR to contribute a deconfliction instruction to the current problem state.
- *gilaco:ProblemDeriveCost* - ask an ILR to evaluate the cost of applying a chosen deconfliction instruction encoded in PSTEPs to the current problem state, where a PSTEP is an atomic action in a partial plan that changes an ACM. Table 1 shows some examples of ACM changes. This problem is only learned by DTLR for now.

The MRE mainly processes system updates caused by posting an instance of *gilaco:Solution* on the blackboard, and continues posting new instances of *gilaco:Problem* until: (1)a maximum cut-off time is reached or (2) all conflicts have been resolved,



initial problem state
Goal: ProblemCharacterizeDeconflictionContext
•current problem state
DTLR
...
CBLR
SPLR
Feature vectors
Goal: ProblemResolveConflict
•current problem state
DTLR
...
CBLR
SPLR
deconfliction-instructions (in Asteps)
Goal: ProblemTranslateAstepToPstep
• deconfliction-instructions (in Asteps)
SPLR
deconfliction-instructions (in Psteps)
Goal: ProblmeDeriveCost
•current problem state
•deconfliction-instructions
Goal: ProblemTry4DConstraint
• current problem state
• deconfliction-instructions
Goal: ProblemTrySaftyConstraint
• current problem state
• deconfliction-instructions
MRE
DTLR
4DCLR
SCLR
cost statements about deconfliction-instructions
new conflicts caused by deconfliction-instructions
violation statements caused by deconfliction-instructions

Figure 5: The ontology-driven process for solving an ACO problem

thereby completing the ACO deconfliction task. Note that conflict detection request is posted on the blackboard via the problem *gilaco:ProblemTry4DConstraint*, and only the 4DCLR has the capability to solve this type of problem.

5 The Meta-Reasoning Executive (MRE)

In both the practice learning phase and the performance phase, the system is required to solve a test problem using the learned knowledge. The Meta-Reasoning Executive (MRE) directs a collaborative performance process during which the ILRs contribute to solving the test problem. This collaborative performance process is modeled as a search for a path from the initial state to a goal state (where the problem is fully solved). The complete solution is a combination of the partial solutions contributed by each ILR.

First, the given test problem is decomposed into a set of subproblems. In general, problem decomposition is a hard task – because the quality of problem solving depends on how the problem is decomposed. In this application, we use the domain knowledge to decompose the original problem. Given an ACO and a set of proposed ACMs, solving the problem consists of removing all existing conflicts. The whole problem is decomposed, and the purpose of each subproblem is to remove one conflict. These subproblems are interrelated, i.e., how a subproblem is solved may affect how others can be solved. Solving one subproblem can also generate new subproblems. The order of solving these subproblems also affects how they can be solved. To manage these interrelationships, the MRE conducts an internal search process, as describe below. In addition, one ILR in particular – the CBLR – learns the priority of these subproblems and provides guidance on the ordering to solve them.

Next, the MRE posts these subproblems on the blackboard, and each ILR then posts its solutions to some of these subproblems. These subproblem solutions are treated as the search operators available at the current state. The MRE then selects one of these subproblem solutions and applies it to the current state, resulting in a new state. New conflicts may appear after applying the subproblem solution. The new state is then evaluated: if it is a goal state (no remaining conflicts), the problem is fully solved; otherwise, the MRE posts all subproblems that correspond to conflicts existing in the new state, and the previous process is repeated.

Figures 6 shows part of the search tree constructed when GILA is performing on Scenario F after learning from the expert demonstration that solves Scenario E (Figures 1 and 2), and practice on Scenario G. The nodes marked with “MREDM-9-XXXX”

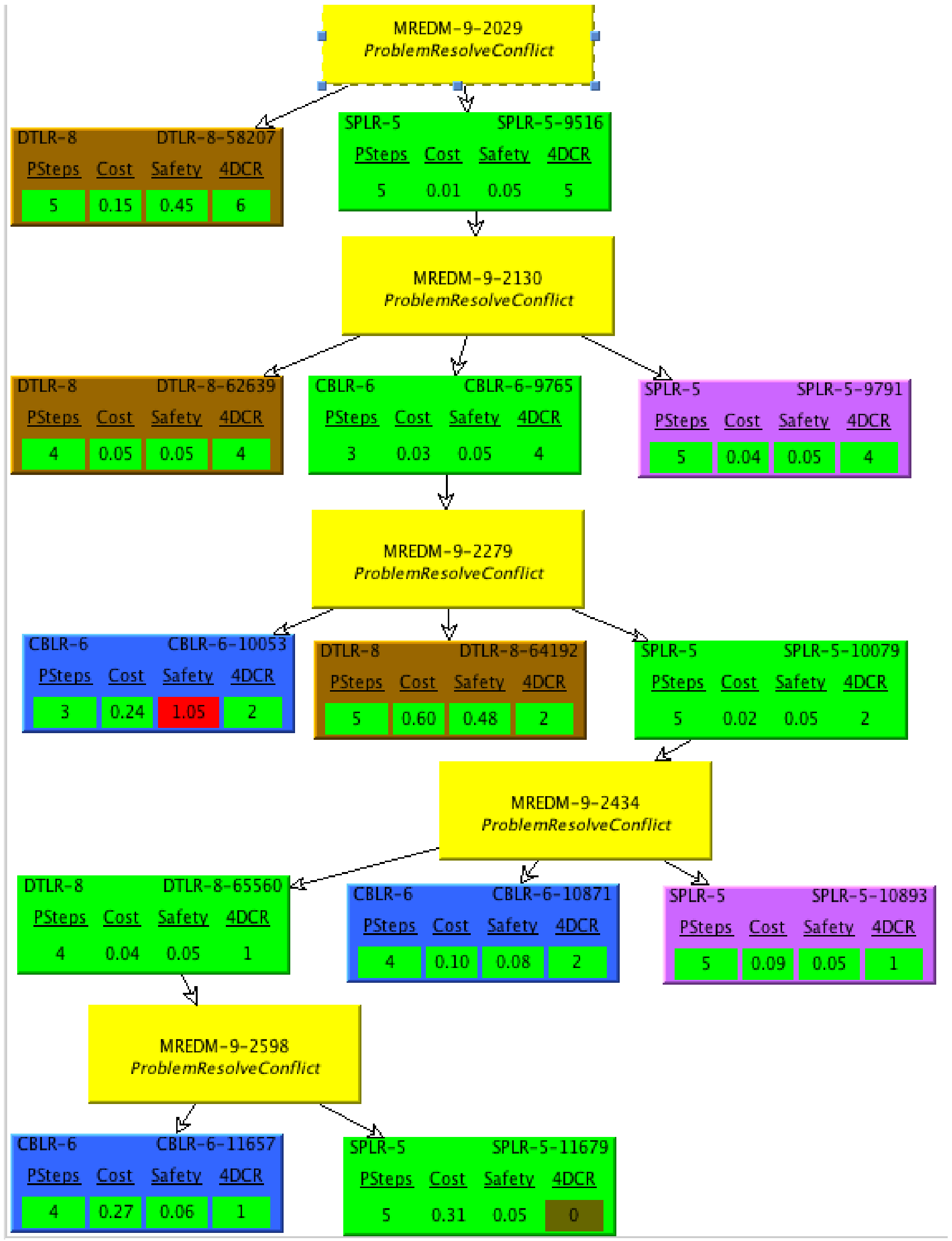


Figure 6: Partial search tree example - GILA performs on Scenario F.

Table 2: Partial execution trace for performance on Scenario F

Step	Execution Trace	Annotation
1	ProblemComputeConflictPriority [elapsed=”01:10:36” id=”MREDM-9-1519” mode=”performance” publisher=”MREDM-9”]	The MRE posts a request to compute the priority for conflicts.
2	SolutionComputeConflictPriority [elapsed=”01:10:36” id=”CBLR-6-6975” mode=”performance” publisher=”CBLR-6” refid=”MREDM-9-1519”]	The CBLR responds with a priority list of all conflicts.
3	ProblemResolveConflict [elapsed=”01:10:37” id=”MREDM-9-1567” mode=”performance” publisher=”MREDM-9”] conflictCount [count=”14”] focusConflict [acm1=”ACM-J-17” acm2=”ACM-J-18” conflictID=”ACM-J-17_ACM-J-18”]	The MRE posts a subproblem to resolve conflicts with the one focus conflict, corresponding tree node: “MREDM-9-1567.”
4	SolutionResolveConflict [elapsed=”01:11:01” id=”CBLR-6-7437” mode=”performance” publisher=”CBLR-6” refid=”MREDM-9-1567”]	The CBLR posts a solution to move the position of ACM-J-18, corresponding tree node: “CBLR-6-7437.”
5	ProblemTrySafetyConstraint [elapsed=”01:11:01” id=”MREDM-9-1585” mode=”performance” publisher=”MREDM-9”] solutionResolveConflict [refid=”CBLR-6-7437”]	The MRE posts a request to check for safety violations in the CBLR solution.
6	SolutionTrySafetyConstraint [elapsed=”01:11:01” id=”SC-4-9899” mode=”performance” publisher=”SC-4” refid=”MREDM-9-1585”]	The Safety Checker responds that the safety violation penalty is 0.0.
7	ProblemDeriveCost [elapsed=”01:11:01” id=”MREDM-9-1589” mode=”performance” publisher=”MREDM-9”]	The MRE posts a request for the execution cost of this CBLR solution.
8	SolutionDeriveCost [elapsed=”01:11:01” id=”DTLR-8-47473” mode=”performance” publisher=”DTLR-8” refid=”MREDM-9-1589”] cost [value=”0.047487963”]	The DTLR responds with the execution cost as 0.05.
9	ProblemTry4DConstraint [elapsed=”01:11:01” id=”MREDM-9-1593” mode=”performance” publisher=”MREDM-9”] solutionResolveConflict [refid=”CBLR-6-7437”]	The MRE posts a request for checking the result after applying the solution proposed by CBLR.
10	SolutionTry4DConstraint [elapsed=”01:11:02” id=”4DR-7-47475” mode=”performance” publisher=”4DR-7” refid=”MREDM-9-1593”] conflictCount [count=”13”]	The 4DCLR responds that the CBLR’s solution resolve one conflict, “ACM-J-17_ACM-J-18,” and no new conflict is introduced; there are 13 remaining conflicts.
11	SolutionResolveConflict [elapsed=”01:11:07” id=”SPLR-5-7467” mode=”performance” publisher=”SPLR-5” refid=”MREDM-9-1567”]	The SPLR posts a solution to move the position of ACM-J-17, with corresponding tree node: “SPLR-5-7467.”
12	ProblemTrySafetyConstraint [elapsed=”01:11:07” id=”MREDM-9-1615” mode=”performance” publisher=”MREDM-9”] solutionResolveConflict [refid=”SPLR-5-7467”]	The MRE posts a request to check for safety violations in the SPLR solution.
13	SolutionTrySafetyConstraint [elapsed=”01:11:07” id=”SC-4-9900” mode=”performance” publisher=”SC-4” refid=”MREDM-9-1615”]	The Safety Checker responds with a safety violation penalty of 0.0.
14	ProblemDeriveCost [elapsed=”01:11:07” id=”MREDM-9-1619” mode=”performance” publisher=”MREDM-9”] solutionResolveConflict [refid=”SPLR-5-7467”]	The MRE posts a request for the execution cost of this SPLR solution.
15	SolutionDeriveCost [elapsed=”01:11:07” id=”DTLR-8-47665” mode=”performance” publisher=”DTLR-8” refid=”MREDM-9-1619”] cost [value=”0.01067276”]	The DTLR responds with an execution cost of 0.05.
16	ProblemTry4DConstraint [elapsed=”01:11:07” id=”MREDM-9-1623” mode=”performance” publisher=”MREDM-9”] solutionResolveConflict [refid=”SPLR-5-7467”]	The MRE posts a request for checking the result after applying the solution proposed by SPLR.
17	SolutionTry4DConstraint [elapsed=”01:11:08” id=”4DR-7-47667” mode=”performance” publisher=”4DR-7” refid=”MREDM-9-1623”] conflictCount [count=”12”]	The 4DCLR responds that the SPLR’s solution resolve two conflicts, “ACM-J-17_ACM-J-18” and “ACM-J-17_ACM-J-19,” and no new conflict is introduced; there are 12 remaining conflicts.
18	ProblemResolveConflict [elapsed=”01:11:09” id=”MREDM-9-1643” mode=”performance” publisher=”MREDM-9”] SolutionResolveConflict [refid=”SPLR-5-7467”] conflictCount [count=”12”] focusConflict [acm1=”ACM-J-15” acm2=”HAVASOUTH” conflictID=”HAVASOUTH_ACM-J-15”]	The MRE selects the SPLR’s solution to explore first, and posts a new subproblem to resolve conflicts with the one focus conflict, corresponding tree node: “MREDM-9-1643.”

represent problem states and the nodes marked with “CBLR-6,” “SPLR-5” or “DTLR-8” represent subproblem solutions (sequences of ACM modifications) posted by the ILRs. The problem state node and subproblem solution nodes alternate. If a problem state node represents the problem state p, and one of its child nodes is a subproblem solution s selected for exploration, a new problem state node is generated representing the result of applying s to p. The ordering of nodes to be explored depends on the search strategy. A best-first search strategy is used in this project. The node with the best evaluation score is selected and explored next.

This search process is directed by the learned knowledge from the ILRs in the following two ways.

First, GILA learns to decide which subproblems to work on initially. It is not efficient to have all ILRs provide solutions to all subproblems, as it takes more time to generate those subproblem solutions and also requires more effort to evaluate them. Because solving one subproblem could make solving the remaining problems easier or more difficult, it is crucial to direct the ILRs to work on subproblems in a facilitating order. This ordering knowledge is learned by the CBLR. In the beginning of the search process, the MRE asks the CBLR to provide a priority ordering of the subproblems (Step 1 in Table 2). Based on this priority list (Step 2 in Table 2), the MRE suggests which subproblem to work on first (listed as the “focus conflict” in Problem Resolve Conflict) (Step 3 and 18 in Table 2). This suggestion is taken by all the ILRs as guidance to generate solutions for subproblems. The ILRs work on the subproblems simultaneously. For instance, Steps 4 and 11 in Table 2 are performed in parallel.

Second, GILA learns to evaluate the proposed subproblem solutions. Each subproblem solution is evaluated using the learned knowledge from the ILRs in the following ways:

1. Check for safety violations in the subproblem solution (Steps 5, 6, 12 and 13 in Table 2). Some subproblem solutions may cause safety violations that make them invalid. The Safety Checker determines whether there is a safety violation and, if yes, how severe the violation is, which is represented by the violation penalty. If the violation penalty is greater than the safety threshold, the MRE discards this subproblem solution. For example, as shown in Figure 6, there is a red box for safety value ”1.05” on Node ”CBLR-6-10053.” The red safety box means that the solution represented by this node has a violation penalty greater than the safety threshold (1.0 in this example); hence, this solution is discarded. Otherwise, if the violation penalty is less than or equal to the safety threshold, the following evaluations are performed.

2. The DTLR derives the execution cost for a subproblem solution that is expected to approximate the expert’s cost function (Steps 7, 8, 14 and 15 in Table 2). The execution cost is learned by the DTLR to measure how close this solution is to the expert’s demonstration. Ideally, GILA is looking for a solution that best mimics the expert, which is a solution with minimal execution cost. However, this cost estimation is not exact due to various assumptions in its learning, such as discretization of the action space and inexact inference.

3. Another ILR, the 4DCLR, performs an internal simulation to investigate the results of applying a subproblem solution to the current state (Steps 9, 10, 16 and 17 in Table 2). The resulting new problem state is evaluated, and the number of remaining conflicts is returned to the MRE as an estimate of how far it is from the goal state, which is the state with no remaining conflicts.

If a subproblem solution does not solve any conflict at all, it is discarded; otherwise, a subproblem solution is evaluated based on the following factors: the cost of executing all subproblem solutions selected on this path ($cumulative\_exec\_cost$), safety violation penalties that would be present if the path were executed ($safety\_penalties$) , and the estimated execution cost and violation penalties of resolving the remaining conflicts ($estimated\_remaining\_cost$). These factors are combined using a linear function with a set of weight parameters:

1. execution.cost.weight (w1)
2. safety.violations.weight (w2)
3. solved.to.remaining.conflict.balance.weight (w3)

The estimation of execution cost and violation penalties for resolving the remaining conflicts is calculated as:

$$estimated\_remaining\_cost = \frac{actual\_cost}{number\_of\_resolved\_conflicts} * number\_of\_remaining\_conflicts$$

$$actual\_cost = w1 * cumulative\_exec\_cost + w2 * safety\_penalties$$

The $estimated\_total\_cost$ is calculated as:

$$estimated\_total\_cost = w3 * actual\_cost + (1 - w3) * estimated\_remaining\_cost$$

The values of the weight parameters $w1$, $w2$, and $w3$ can be varied to generate different search behaviors. Based on the $estimated\_total\_cost$, the MRE determines the ordering of nodes to be explored (Step 18 in Table 2). The search process stops when a goal state is reached, i.e., when there are no remaining conflicts. In Figure 6, there is a brown box with the 4DCR value of ”0” on Node ”SPLR-5-11679.” This node represents a goal state because the 4DCLR reports that the number of remaining conflicts in the current state is ”0.”

6 Integrated Learning Reasoning systems (ILRs)

The GILA system consists of four different ILRs, and each learns in a unique way. The symbolic planner learner-reasoner (SPLR) learns decision rules and value functions, the decision-theoretic learner-reasoner (DTLR) learns a linear cost function, the case-based learner-reasoner (CBLR) learns and stores a feature-based case database, and the 4D-deconfliction and constraint learner-reasoner (4DCLR) learns hard constraints on the schedules in the form of rules. In this section, we describe the internal knowledge representation formats and learning/reasoning mechanisms of these ILRs and how they work inside the GILA system.

6.1 The Symbolic Planner Learner-Reasoner (SPLR)

Large portions of human strategies can be compactly represented as reactive control rules. By reactive control rules, we mean rules that suggest which action to take in reaction to the current situation. For example, when we travel from Chicago to New York, the simplest strategy is the following. First, if one is in Chicago, “Go to the airport.” Second, if one is already at the airport, “Take the plane.” This example shows that a simple but useful strategy can be very reactive. That is, in this travel example, we take actions conditional on the current situation. In fact, in many situations, humans have simple reactive strategies. This also seems to be the case for experts in the airspace deconfliction problem. Experts’ deconfliction strategies are predominantly reactive, conditional on properties of the conflicts.

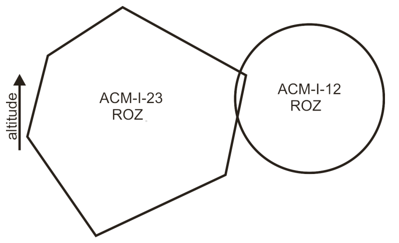
Figure 7 shows three representative deconfliction strategies. A conflict in airspace is 4D. In other words, if any one of the four dimensions – longitude, latitude, altitude or time – is in conflict, then the two airspaces conflict. The conflict is removed once each of these 4D conflicts is resolved. In Figure 7, each strategy corresponds to the removal of one or two of the dimensions’ conflicts. The leftmost figure shows the elimination of the conflict by changing the altitude. The center figure shows a time change, and the rightmost figure shows geographical movement. In general, most geometric deconfliction changes involve both longitude and latitude. This is natural since the combination of the two is likely to have shorter Euclidean distance than changing only one of them. Experts do occasionally change multiple dimensions simultaneously, e.g., by changing both altitude and time at the same time.

A deconfliction expert’s strategy is typically reactive. The expert’s most important decision usually involves mission types that are in conflict. Circumstantial facts are used too, e.g., what missions exist for the airspaces surrounding the conflict. If we look at the missions in detail, we can understand why mission-aware reactive deconfliction is sensible. For example, to resolve a conflict involving a missile campaign, we cannot change the geometry of the missile campaign mission. A missile campaign targets an exact enemy location, and therefore the destination is fixed. Thus, when missile corridors are in conflict, experts frequently attempt to slide the time to remove the conflict. As long as the missiles are delivered to the target, shifting time by a small amount may not compromise the mission objective. Though natural for a human, reasoning of this type is hard for a machine – unless we provide a carefully coordinated knowledge base. Rather, it may be easier for a machine to learn to “change time” *reactively* when the “missile campaign” is in conflict. From the subject matter expert’s demonstration, the machine can learn what type of deconfliction strategy was used in which types of missions. With a suitable general relational language, the system can learn good reactive deconfliction strategies (Yoon, Fern, and Givan 2002; Yoon and Kambhampati 2007; Khardon 1999; Martin and Geffner 2000).

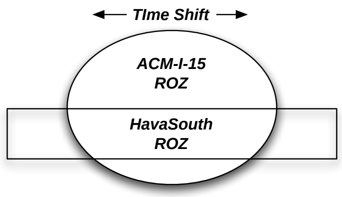
Of course, if all of the experts’ strategies were so simple, a straightforward rule-based expert system could express them. Unfortunately, the airspace management decision-making process can be quite complex and there are always exceptions. In particular, when multiple missions collide in a very tight airspace, the expert must compare the importance of the impending missions, as well as other related issues, all of which are quite challenging to operationalize in a system. Therefore, the SPLR’s objective is to try to capture the reactive strategy that covers a majority of the situations, rather than the complete expert’s strategy. The SPLR’s learned reactive rules are thus approximately, rather than precisely, correct.

6.1.1 Airspace Deconfliction Problem Example

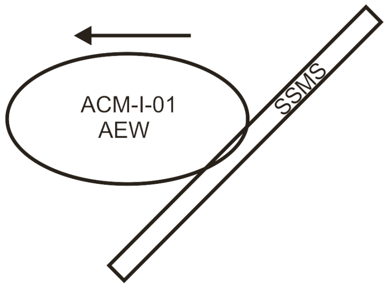
A fundamental question addressed by this project is how to provide the machine with a compact language system that captures an expert’s strategy without human knowledge engineering. The SPLR’s contribution to addressing this question is its language bias, which is very appropriate for the airspace deconfliction problem, and has also been used successfully in other, different applications. This language bias is based on an ontology that describes the airspace deconfliction problem with relational symbols. In particular, the SPLR automatically enumerates its strategies in our formal language system, and finds the best strategy. In the following, we describe the SPLR’s language bias in detail, including its syntax and semantics. We call the learned strategy a



(a) The expert removes the conflict by changing the altitude of ACM-I-23. The expert changes the altitude for "ROZ" usage when its shape is "Polygon" and it conflicts with another "ROZ" ACM.



(b) The expert shifts the time when "orbit" type "ROZ" ACM is in conflict with another "ROZ" ACM in "corridor" shape.



(c) The expert moves an ACM of "AEW" usage when it conflicts with an ACM of "SSMS" usage.

Figure 7: Expert deconfliction examples. (ROZ:Restricted Operations Zone. AEW: Airborne Early Warning Area. SSMS: Surface-to-Surface Missile System)

"direct policy representation," where this terminology is borrowed from the literature on Markov Decision Processes (Yoon, Fern, and Givan 2002; Yoon and Kambhampati 2007; Khardon 1999; Martin and Geffner 2000).

6.1.2 Learning Relational Symbolic Actions

Policy Representation

The relational action selection strategy is represented with a decision list (Rivest 1987). A decision list DL consists of ordered rules r. In our approach, a DL outputs "true" or "false" after receiving an input action. The DL's output is the disjunction of each rule's outputs, $\bigvee r$. Each rule r consists of binary features F_r, and a conjunction ($\bigwedge F_r$) of the features is the result of the rule succeeding the action. A binary feature F_r outputs "true" or "false" after receiving an input action.

To decide which action to take, the SPLR considers all the actions available in the current situation. If there is no action with "yes" from the DL, it chooses a random action. Among the actions with "yes," it takes the action that has the earliest rule r to output "yes." Ties are broken randomly if there are multiple actions with "yes" from the same earliest rule. Next, we describe the features for the action.

Features

A state consists of a set of ground-truth facts of first-order logic. For example, a relation set: S = (use ACM-I-01 AEW), (conflict ACM-I-01 ACM-I-18), (use ACM-I-18 SSMS). . . describes some part of the airspace deconfliction problem state. The description expresses that the use of airspace ACM-I-18 is "SSMS" (Surface-to-Surface Missile System) and it is in conflict with ACM-I-01. See Figure 7(c). The binary feature $F = (A \; i \; C)$ for action specifies constraints on the action type A, ("move," "altitude," etc.) and constraints on the argument i in C. The feature is true if all the constraints are satisfied. Constraints on the action type are straightforward and we detail the constraints on the arguments. To do this, we introduce Taxonomic syntax (McAllester and Givan 1993). Variable-free taxonomic syntax is a convenient tool for relational representation (Yoon, Fern, and Givan 2002). The syntax for a taxonomic expression C is defined as follows.

$$C = (R, C_1, ..., C_{i-1}, ?, C_{i+1}, ..., C_n(R))$$

R is the relational symbol. For example, "use" in (use ACMB-1 SSMS) is a relational symbol. $n(R)$ is the arity function for the relational symbol R. ? is the argument position that is intentionally left void and is the set of objects described by the expression C in the equation above. In other words, the equation describes the set of objects that makes the relation R true at position ? once other positions are in corresponding class expressions. An example of a taxonomic expression $C = (use \; ? \; SSMS)$ is {ACM-I-18} for the ongoing example S in the above paragraph and in Figure 7(c), since ACM-I-18 makes "(use ? SSMS)" true when it is in the ? position in the state S. For a more detailed description of taxonomic syntax, see (Yoon, Fern, and Givan 2002). Note that a limited set of taxonomic expressions can easily be enumerated from the domain definition in the GILA ontology. With the taxonomic syntax, we define binary features F as:

$$F = (A \; i \; C)$$

The feature F shows that the action type is A and the argument i is in C, which is a taxonomic expression. The feature is a binary feature for an action. It becomes true for an action when the type of the action is A and the ith argument belongs to C. For example, (move 0 (use ? AEW)) is a feature in the airspace deconfliction domain. The feature is true for actions with type "move" and the 0th argument is in class expression (use ? AEW), which designates airspaces of usage "AEW" (Airborne Early Warning Area). This feature is true for action "(move ACM-I-01)" for the example S in Figure 7(c). The feature semantically expresses that geometric movement is favored when the use of the airspace is "AEW." The binary features can also be automatically enumerated for the given set of taxonomic expressions, and therefore the whole feature set can be enumerated automatically from the domain definition.

The Learning

The learning of a direct policy with relational actions is now straightforward. The expert's deconfliction actions, e.g., *move*, are the training examples. Given a demonstration trace, each action is turned into a set of binary values, which is then evaluated against pre-enumerated binary features as described above. We used a Rivest-style decision list learning package implemented as a variation of PRISM (Cendrowska 1987) from the Weka Java library. The plain PRISM algorithm cannot cover negative examples, but our variation allows for such coverage. For the expert's selected actions, the SPLR constructs rules with "true" binary features when negative examples are the actions that were available but not selected. After a rule is constructed, examples explained (i.e., covered) by the rule are eliminated. The learning continues until there are no training examples left. We list the empirically learned direct policy example from Figure 7(a), 7(b) and 7(c) in the following:

1. (altitude 0 (Shape ? Polygon)) & (altitude 0 (Conflict ? (Shape ? Circle))) : Learned from Figure 7(a)
2. (time 0 (use ? ROZ)) & (time 0 (Conflict ? (Shape ? Corridor))) : Learned from Figure 7(b)
3. (move 0 (use ? AEW)) & (move 0 (Conflict ? (use ? SSMS))) : Learned from Figure 7(c)

Other Learning Components in the SPLR

We have developed two more learning components in the SPLR. These components are based on our novel feature system, but with slight variations.

First, besides choosing a strategic deconfliction action, specific values must be assigned. If we opted to change the time, by how much should it be changed? Should we impose some margin beyond the deconfliction point? How much margin is good enough? To answer these questions, we consulted the expert demonstration trace, which has records concerning the margin. We used linear regression to fit the observed margins. The features used for this regression fit were the same as those used for direct policy learning; thus, feature values are boolean. The intuition is that the margins generally depend on the mission type. For example, missiles need a narrow margin because they must maintain a precise trajectory. The margin representation is a linear combination of the features, e.g., Move Margin = $\sum w_i \times F_i$ (margin of move action). We learned weights w_i with linear regression.

Second, experts emphasize order when addressing conflicts; thus conflicts are given a priority order during performance. Given a set of conflicts, the conflicts are ranked with decision tree learning. First, experts show the order of conflicts. Then, in our learning scheme, each pair of conflicts is used as a training example. An example is true if the first member of a pair is given priority over the second. After learning, for each pair of conflicts (c_1, c_2), the learned decision tree answers "true" or "false." "True" means that the conflict c_1 has higher priority. The rank of a conflict is the sum of "true" values against all the other conflicts, with ties broken randomly. The features for the decision tree were again from the similar feature set. Instead of action type, we used (Conflict ? C) as a feature, meaning airspaces in ? are in conflict with a class of C. In this example variation, (Conflict ? (use ? SSMS)) describes a conflict involving airspace use of SSMS. The rank of a conflict is primarily determined by the missions involved. For example, two conflicting missile corridors are given a high priority due to their sensitivity to change.

6.2 The Decision Theoretic Learner-Reasoner (DTLR)

The *Decision-Theoretic Learner-Reasoner* (DTLR) learns a cost function over possible solutions to problems. It is assumed that the expert's solution optimizes the cost function subject to some constraints. The goal of the DTLR is to learn a close approximation of the expert's cost function. We pose the problem of learning the cost function as an instance of structured prediction (Bakir et al. 2007). Once the cost function is learned, the performance algorithm of the DTLR uses this function to try to find a minimal cost solution with iterative-deepening search.

6.2.1 Learning a Cost Function via Structured Prediction

We will formalize the cost function learning in the framework of structured prediction. We define a structured prediction problem as a tuple $\{\mathcal{X}, \mathcal{Y}, \Psi, L\}$, where $\mathcal{X}$ is the *input space* and $\mathcal{Y}$ is the output space. We are aided in the learning process by the *joint feature function* $\Psi : \mathcal{X} \times \mathcal{Y} \mapsto \Re^n$ which defines the joint features on both inputs and outputs. The *loss function*, $L : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \mapsto \Re$, quantifies the relative preference of two outputs given some input. Formally, for an input $\mathbf{x}$ and two outputs $\mathbf{y}$ and $\mathbf{y}'$, $L(\mathbf{x}, \mathbf{y}, \mathbf{y}') > 0$ if $\mathbf{y}$ is a better choice than $\mathbf{y}'$ given input $\mathbf{x}$ and $L(\mathbf{x}, \mathbf{y}, \mathbf{y}') \leq 0$ otherwise. We use a margin-based loss function used in the *logitboost* procedure (Friedman, Hastie, and Tibshirani 1998) and is defined as $L(\mathbf{x}, \mathbf{y}, \mathbf{y}') = log\left(1 + e^{-m}\right)$, where m is the margin of the training example (see Section 6.2.2 for details).

The decision surface is defined by a linear scoring function over the joint features $\Psi(\mathbf{x}, \mathbf{y})$ given by the inner product $\langle \Psi(\mathbf{x}, \mathbf{y}), \mathbf{w} \rangle$, where $\mathbf{w}$ is the vector of learned model parameters, and the best $\mathbf{y}$ for any input $\mathbf{x}$ has the highest score. The specific goal of learning is, then, to find $\mathbf{w}$ such that $\forall i : \text{argmax}_{\mathbf{y} \in \mathcal{Y}} \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w} \rangle = \mathbf{y}_i$.

In the case of ACO scheduling, an input drawn from this space is a combination of ACO and a deconflicted ACM to be scheduled. An output $\mathbf{y}$ drawn from the output space $\mathcal{Y}$ is a schedule of the deconflicted ACM. The joint features are x-y coordinate change, altitude change and time change for each ACM, and other features such as changes in the number of intersections of the flight paths with the enemy territory.

6.2.2 Structured Gradient Boosting (SGB) for Learning a Cost Function

We now describe our *Structured Gradient Boosting* (SGB) algorithm (Parker, Fern, and Tadepalli 2006), which is a gradient descent approach to solving the structured prediction problem. We will make use of a notion of margin for the structured prediction problem. Suppose that we have some training example $\mathbf{x}_i \in \mathcal{X}$ with the correct output $\mathbf{y}_i \in \mathcal{Y}$. We define $\widehat{\mathbf{y}}_i$ to be the best incorrect output for $\mathbf{x}_i$ according to the current model parameters. That is,

$$\widehat{\mathbf{y}}_i = \text{argmax}_{\mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i} \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w} \rangle$$

We define the *margin* to be the amount by which the model prefers $\mathbf{y}_i$ to $\widehat{\mathbf{y}}_i$ as a output for $\mathbf{x}_i$. The margin m_i for a given training example is then $m_i = \langle \delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i), \mathbf{w} \rangle$, where $\delta\left(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i\right)$ is the difference between the feature vectors $\Psi\left(\mathbf{x}_i, \mathbf{y}_i\right)$ and $\Psi\left(\mathbf{x}_i, \widehat{\mathbf{y}}_i\right)$. The margin determines the loss as shown in Step 2 of the pseudo code of Table **??**. The parameters of the cost function are adjusted to reduce the gradient of the cumulative loss over the training data (see (Parker, Fern, and Tadepalli 2006) for more details).

The problem of finding $\widehat{\mathbf{y}}_i$, which is encountered during both learning and performance, is called the *Argmax* problem. We define a discretized space of operators, namely, the altitude, time, radius and x-y coordinates, based on the domain knowledge, to produce various possible plans to deconflict each ACM. A simulator is used to understand the effect of a deconfliction plan. Based on their potential effects, we evaluate these plans using the model parameters and find the best scoring plan that resolves the conflict. Exhaustive search in this operator space would be optimal for producing high quality solutions, but has excessive computational cost. We use Iterative Deepening Depth First (IDDFS) search by considering single changes before multiple changes, and smaller amounts of changes before larger amounts of changes, thereby trading off the quality of the solution with the search time.

Algorithm 3 STRUCTURED GRADIENT BOOSTING
Input: $\{\langle x_i, y_i \rangle\}$ the set of training examples
B the number of boosting iterations
Output: weights of the cost function $\mathbf{w}$

```
 1: Initialize the weights w = 0
 2: repeat
 3:     for each training example $(x_i, y_i)$ do
 4:         Solve Argmax: $\widehat{\mathbf{y}}_i = \text{argmax}_{\mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i} \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w} \rangle$
 5:         Compute Training loss: $L(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i) = \log(1 + e^{-m_i})$, where $m_i = \langle \delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i), \mathbf{w} \rangle$
 6:     end for
 7:     Compute Cumulative training loss: $L = \sum_{i=1}^{n} L(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i)$
 8:     find the gradient of the cumulative training loss $\nabla L$
 9:     Update weights: $\mathbf{w} = \mathbf{w} - \alpha \nabla L$
10: until convergence or $B$ iterations
11: return the weights of the learned cost function w
```

6.2.3 Illustration of Gradient Boosting

We will geometrically illustrate our gradient boosting algorithm with the help of Figure 8. Suppose that we have four training examples $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ and (x_4, y_4). As explained in the previous section, our features depend on both the input x and the output y, i.e., $\Psi(x, y)$. We represent the data points corresponding to features $\Psi(x_i, y_i)$ of the training examples x_i with respect to their true outputs y_i with $\oplus$, i.e., positive examples and data points corresponding to features $\Psi(x_i, \hat{y}_i)$ of the training examples x_i with respect to their best scoring outputs $\hat{y}_i$ with $\ominus$, i.e., negative examples. Note that the location of the positive points do not change, unlike the negative points whose locations change from one iteration to another, i.e., the best scoring negative outputs $\hat{y}_i$ change with the weights, and hence the feature vectors of the negative examples $\Psi(x_i, \hat{y}_i)$ change. We show three boosting iterations of our algorithm, one row per iteration. In each row, the left figure shows the current hyperplane (cost function) along with the negative examples according to the current cost function and the right figure shows the cost function obtained after updating the weights in a direction that minimizes the cumulative loss over all training examples, i.e., a hyperplane that separates the positive examples from negative ones (if such a hyperplane exists). As the boosting iterations increase, our cost function is moving towards the true cost function and it will eventually converge to the true cost function.

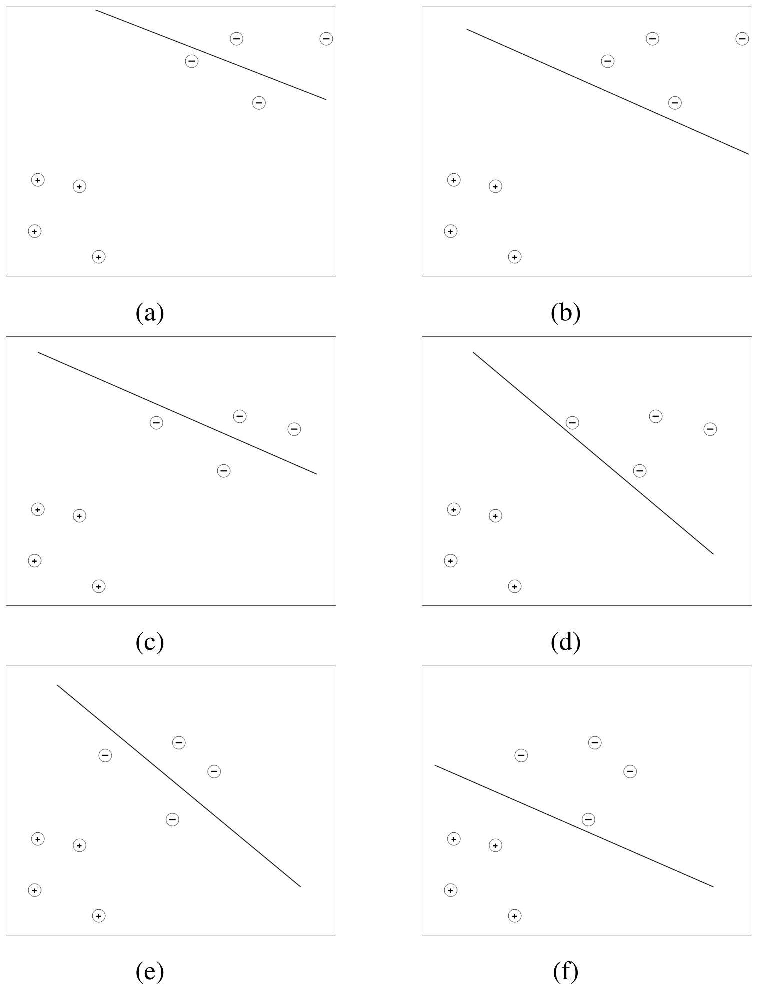


Figure 8: In all of these figures, $\oplus$ corresponds to the feature vectors of the training examples with respect to their true outputs, $\ominus$ corresponds to the feature vectors of the training examples with respect to their best scoring negative outputs and the separating hyperplane corresponds to the cost function. (a) Initial cost function and negative examples before 1st iteration (b) Cost function after 1st boosting iteration (c) Cost function and negative examples before 2nd iteration (d) Cost function after 2nd boosting iteration (e) Cost function and negative examples before 3rd iteration (f) Cost function after 3rd boosting iteration

6.2.4 Relation to Other Structured Prediction Algorithms

Our gradient boosting algorithm is closely related to two online algorithms - *Structured perceptron* (Collins 2002) and *Structured MIRA* (McDonald, Crammer, and Pereira 2005). Structured perceptron is an extension of the standard perceptron algorithm to the structured prediction setting. At each training iteration t, for each training example $\mathbf{x}_i$, the learner predicts the best scoring output $\widehat{\mathbf{y}}_i$ using the current weights w_t. If the predicted output $\widehat{\mathbf{y}}_i$ is not equal to the true output $\mathbf{y}_i$, then the weights are updated as shown below:

$$w_{t+1} = w_t + \Psi(\mathbf{x}_i, \mathbf{y}_i) - \Psi(\mathbf{x}_i, \widehat{\mathbf{y}}_i)$$

Structured MIRA is the structured version of the online passive-aggressive algorithm MIRA (Margin Infused Relaxed Algorithm). This algorithm always tries to maintain a minimum margin threshold (η) between the true output $\mathbf{y}_i$ and the predicted best scoring output $\widehat{\mathbf{y}}_i$ using the current weights w_t. The weights are updated to the solution of the following quadratic programming (QP) problem:

$$\begin{aligned} w_{t+1} = & \min_{w} \frac{1}{2}||w - w_t||^2 \\ & \text{s.t. } w \cdot \Psi(\mathbf{x}_i, \mathbf{y}_i) - w \cdot \Psi(\mathbf{x}_i, \widehat{\mathbf{y}}_i) \geq \eta \end{aligned}$$

There is a closed form solution to the above QP problem that is very similar to the MIRA update. Both of these algorithms are purely online, i.e., weight updates are based on a single training example. On the other hand, weight updates in our gradient boosting algorithm are based on the cumulative loss of all the training examples.

SVM-Struct (Tsochantaridis et al. 2005) and Max-Margin Markov Networks (Taskar, Guestrin, and Koller 2004) are two other batch algorithms that are closely related to our algorithm. To start with, both SVM-Struct and M^3 Nets are formulated as a QP problem with an exponential number of constraints. M^3 Nets uses the structure of a graphical model to cleverly factor the exponential number of constraints into a polynomial number of marginal constraints. However, this reduced set also includes equality constraints and slack variables, which need careful design and tuning in quadratic solvers. On the other hand, SVM-Struct uses a cutting-plane algorithm to approximate the solution to this hard QP problem. SVM-Struct starts with an empty set of constraints, iteratively adds the most violated constraint for each training example to the constraint set, and solves the optimization problem subject to this set of constraints. It repeats this process until there are no additions to the constraint set, for all the training examples. In contrast, gradient boosting does not carry any negative examples from the previous boosting iteration to the next one. This could sometimes lead to zigzag effects, which are mitigated by adjusting the weights based on a set of training examples. The advantage of SVM-Struct or M^3 Nets comes at the cost of huge training times, whereas gradient boosting converges to a good weight vector with a small number of iterations, thereby making it a good choice for time-constrained applications.

6.2.5 What Kind of Knowledge Does the DTLR Learn?

We explain, through an example case E-G-F[1], what was learned by the DTLR from the expert's demonstration and how the knowledge was applied while solving problems during performance mode. Before training, weights of the cost function are initialized to zero. For each ACM that was moved to resolve conflicts, we create a training example for our gradient boosting algorithm. The expert's plan that deconflicts the problem ACM corresponds to the correct solution for each of these training examples. Learning is done using the gradient boosting algorithm described before, by identifying the best possible wrong solution and computing a gradient update towards the correct solution. For example, in the expert's demonstration (Scenario E, shown in Table 1) which was used for training, all the deconflicting plans are either altitude changes or x-y coordinate changes. Hence, the DTLR learns a cost function that prefers altitude and x-y coordinate changes to time and radius changes.

During performance mode, when given a new conflict to resolve, the DTLR first tries to find a set of deconflicting plans using Iterative Deepening Depth First (IDDFS) search. Then, it evaluates each of these plans using the learned cost function and returns the plan with minimum cost. For example, in Scenario F (which was used for performance mode), when trying to resolve conflicts for *ACM-J-15*, it found six plans with a single (minimum altitude) change and preferred the one with minimum change by choosing to increase the minimum altitude by 2000 (as shown on row #9 in Table 3).

6.3 The Case-Based Learner-Reasoner (CBLR)

Case-based reasoning (CBR) (Aamodt and Plaza 1994) consists of solving new problems by reasoning about past experience. Experience in CBR is retained as a collection of *cases* stored in a case library. Each of these cases contains a past problem and the associated solution. Solving new problems involves identifying relevant cases from the case library and reusing or adapting their solutions to the problem at hand. To perform this adaptation process, some CBR systems, such as the CBLR, require additional adaptation knowledge.

In the CBLR module, the process of case-based learning is performed by observing an expert solving problems, extracting the problem descriptions, features and solutions, and then storing them as cases in a case library. In GILA, the CBLR uses several specialized case libraries, one for each type of problem that the CBLR can solve. For each of these libraries, the CBLR has two learning modules: one capable of learning cases and one for adaptation knowledge. Adaptation knowledge in the CBLR is expressed as a set of transformation rules and a set of constraints. Adaptation rules capture how to transform the solution from the retrieved case to solve the problem at hand, and the constraints specify the combinations of values that are permitted in the solutions being generated.

[1]E-G-F means using Scenario E (Figures 1 and 2) for demonstration, G for practice and F for performance; see Table 4 for more details.

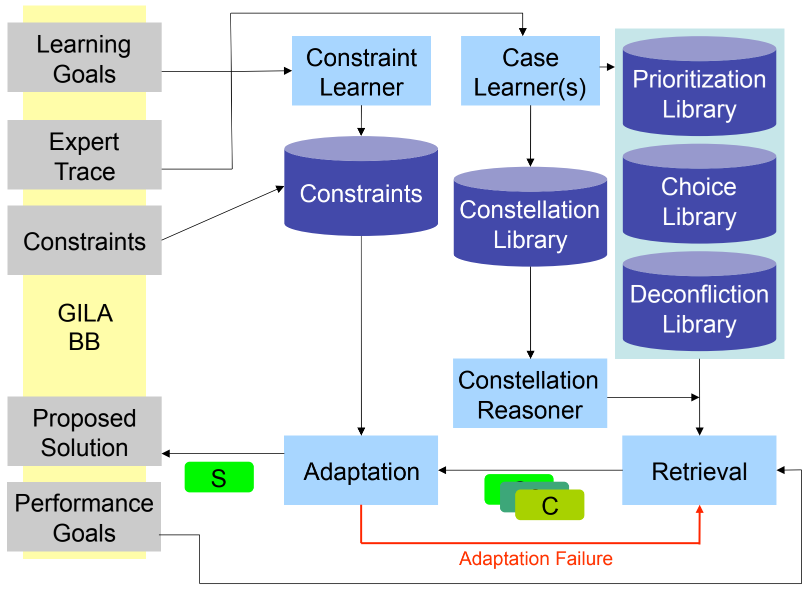


Figure 9: The architecture of the Case-Based Learner-Reasoner

Figure 9 shows the overall architecture of the CBLR, including the connections among components and the GILA blackboard. Specifically, the CBLR contains the following case libraries. It uses a *Prioritization Library*, which contains a set of cases for reasoning about the priority, or order, in which conflicts should be solved. It contains a *Choice Library*, which, given a conflict between two ACMs, is used to determine which ACM will be moved. Finally, it contains a *Constellation Library* and a *Deconfliction Library*, which are used within a hierarchical process. The *Constellation Library* is used by the CBLR to characterize the neighborhood surrounding a conflict. The neighborhood provides information that is then used to help retrieve cases from the *Deconfliction Library*.

6.3.1 Learning in the CBLR

Each case library contains a specialized case learner, which learns cases by extracting them from an expert trace. Each case contains a problem description and an associated solution. Figure 10 shows sample cases learned from the expert trace.

Prioritization

Using information from the expert trace available to the CBLR, the Priority Learner constructs prioritization cases by capturing the order in which the expert prioritizes the conflicts in the trace. From this, the CBLR learns prioritization cases, storing one case per conflict. Each case contains a description of the conflict, indexed by its features, along with the priority assigned by the expert. These cases are used by the CBLR to provide the priority order of deconfliction to the MRE. The order in which conflicts are resolved can have a significant impact on the quality of the resulting airspace.

Choice

Given a conflict between two ACMs, the CBLR uses the Choice Case Library to store the identifier of the ACM that the expert chose to modify. Each time the expert solves a conflict in the trace, the CBLR learns a choice case. The solution stored with the case is the ACM that is chosen to be moved. The two conflicting ACMs and the description of the conflict are stored as the problem description for the case.

Hierarchical Deconfliction

The CBLR uses a hierarchical reasoning process to solve a conflict using a two-phased approach. The first phase determines what method an expert is likely to use when solving a deconfliction problem. It does this by describing the "constellation" of the conflict. "Constellation" refers to the neighborhood of airspaces surrounding a conflict. The choices for deconfliction available to an airspace manager are constrained by the neighborhood of airspaces, and the Constellation Case Library allows the CBLR to mimic this part of an expert's decision-making process. A constellation case consists of a set of features that characterize the degree of congestion in each dimension (latitude-longitude, altitude and time) of the airspace. The solution stored is the dimension within which the expert moved the ACM for deconfliction (e.g., change in altitude, orientation, rotation, radius change, etc.).

The second CBLR phase uses the Deconfliction Case Library to resolve conflicts once the deconfliction method has been determined. A deconfliction case is built from the expert trace by extracting the features of each ACM, as shown below.

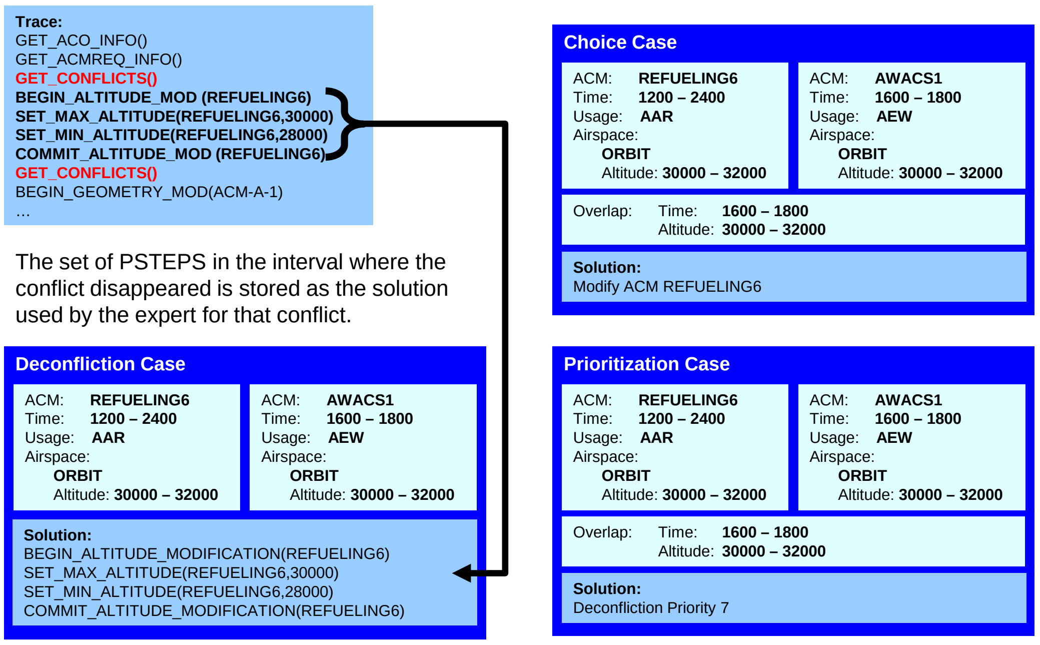


Figure 10: A set of sample cases in CBLR, learned from a subset of PSTEPs from the expert trace.

- ACO, ACMREQ, and ACM Identifiers
- ACM Start and Stop Times
- ACM Usage (e.g. AAR)
- ACM Airspace Shape (e.g. orbit, corridor)
- ACM Minimum and Maximum Altitudes
- ACM Airspace Geometry (full XML description)

This set of domain-specific features was chosen manually based on the decision criteria of human experts. In addition to these two sets of features (one for each of the two conflicts in a pair), a deconfliction case includes a description of the overlap, as shown below:

- Altitude Up/Down - the altitude change required of ACM1 to deconflict
- Time Earlier/Later - the time change required of ACM1 to deconflict
- XY North/South/East/West - the geographical change required of ACM1 to deconflict

The solution is a set of PSTEPs[2] that describe the changes to the ACM, as illustrated in Figure 10. These PSTEPs represent the changes that the expert made to the chosen ACM in order to resolve the conflict. Whereas the constellation phase determines an expert's likely deconfliction method, the deconfliction phase uses that discovered method as a highly-weighted feature when searching for the most appropriate case in the deconfliction case library. It then retrieves the most similar case based on the overall set of features and adapts that case's solution to the new deconfliction problem. We refer to this two-phase process as *hierarchical deconfliction*.

In order to learn adaptation knowledge, the CBLR uses transformational plan adaptation (Muñoz-Avila and Cox 2007) to adapt deconfliction strategies, using a combination of adaptation rules and constraints. Adaptation rules are built into the CBLR. This

[2]Recall from above that a PSTEP is an atomic action in a partial plan that changes an ACM.

rule set consist of five common-sense rules that are used to apply the previously successful deconfliction solution from one conflict to a new problem. For example, "If the overlap in a particular dimension between two airspaces is X, and the expert moved one of them X+Y units in that dimension, then in the adapted PSTEP we should compute the overlap Z and move the space Z+Y units." If more than one rule is applicable to adapt one PSTEP, the adaptation module will propose several candidate adaptations, as explained later.

During the learning process, one challenge in extracting cases from a demonstration trace involves the identification of the sequence of steps that constitutes the solution to a particular conflict. The expert executes steps using a human-centric interface, but the resulting sequence of raw steps, which is used by the ILRs for learning, does not indicate which steps apply to which conflict. The CBLR overcomes this limitation by executing each step in sequence, comparing the conflict list before and after each step to determine if a conflict was resolved by that step.

Constraints are learned both from the expert trace and from other GILA modules (such as the 4DCLR). To learn constraints, the CBLR evaluates the range of values that the expert permits. For instance, if the expert sets all altitudes of a particular type of aircraft to some value between 10,000 and 30,000 feet, the CBLR will learn a constraint that limits the altitude of aircraft type to a minimum of 10,000 feet and a maximum of 30,000 feet. These simple constraints are learned automatically by a constraint learning module inside the CBLR. This built-in constraint learner makes performance more efficient by reducing the dependence of the CBLR on other modules. However, another module in GILA (4DCLR) is able to learn more complex and accurate constraints, which are posted to the blackboard of GILA, and these constraints are used by the CBLR to enhance the adaptation of its solutions.

6.3.2 Problem Solving in the CBLR

The CBLR uses all the knowledge it has learned (and stored in the multiple case libraries) to solve the airspace deconfliction problems. The CBLR is able to solve a range of problems posted by the MRE. For each of the case-retrieval processes, the CBLR uses a weighted Euclidean distance to determine which cases are most similar to the problem at hand. The weights assigned to each feature are based on the decision-making criteria of human experts.

During the performance phase of GILA, a prioritization problem is sent by the MRE that includes a list of conflicts and asks the ILRs to rank them by priority. The assigned priorities will determine the order in which conflicts will be solved by the system. To solve one such problem, the CBLR first assigns priorities to all the conflicts in the problem by assigning each conflict the priority of the most similar priority case in the library. After that, the conflicts are ranked by priority (and ties are solved randomly).

Next, a deconfliction problem is presented to each of the ILRs so that they can provide a solution to a single conflict. The CBLR responds to this request by producing a list of PSTEPs. It solves a conflict using a three-step process. First, it decides which ACM to move using the choice library. It then retrieves the closest match from the Constellation Library and uses the associated solution as a feature when retrieving cases from the Deconfliction Library. It retrieves and adapts the closest n cases (where $n = 3$ in our experiments) to produce candidate solutions. It tests each candidate solution by sending it to the 4DCLR module, which simulates the application of that solution. The CBLR evaluates the results and selects the best solutions, that is, those that solve the target conflict with the lowest cost. A subset of selected solutions is sent to the MRE as the CBLR's solutions to the conflict.

Adaptation is only required for deconfliction problems, and is applied to the solution retrieved from the deconfliction library. The process for adapting a particular solution S, where S is a list of PSTEPs, as shown in Figure 10, works as follows:

1. Individual PSTEP adaptation: Each individual PSTEP in the solution S is adapted using the adaptation rules. This generates a set of candidate adapted solutions AS.
2. Individual PSTEP constraint checking: Each of the PSTEPs in the solutions in AS is modified to comply with all the constraints known by the CBLR.
3. Global solution adaptation: Adaptation rules that apply to groups of PSTEPs instead of to individual PSTEPs are applied; some unnecessary PSTEPs may be deleted and missing PSTEPs may be added.

6.3.3 CBLR Results

The GILA system was developed to further machine-learning research. During GILA development, we were required to minimize encoded domain knowledge and maximize machine learning. One strength of the case-based learning approach is that it learns to solve problems in exactly the same way that the expert does, with very little pre-existing knowledge. During this project, we performed "Garbage-in/Garbage-out" (GIGO) experiments that tested the learning nature of each module by teaching the system with incorrect approaches, then testing the modules to confirm that they used the incorrect methods during performance. This technique was designed to test that the ILR used knowledge that was learned rather than encoded. The case-based learning approach successfully learned these incorrect methods and applied them in performance mode. This proves that the CBLR learns from the expert trace, performing and executing very much like the expert does. This also allows the CBLR to learn unexpected solution approaches when they are provided by an expert. If such a system were to be transitioned with a focus on deconfliction performance (rather than machine learning performance), domain knowledge would likely be included.

The CBLR also responded very well to incremental learning tests. In these tests the system was taught one approach to problem solving at a time. When the system was taught to solve problems using only altitude changes, the CBLR responded in performance mode by attempting to solve all problems with altitude changes. When the system was taught to solve problems by making geometric changes, the CBLR responded in performance mode by using both of these methods to solve problems, confirming that the CBLR's problem-solving knowledge was learned rather than being previously stored as domain knowledge.

Moreover, the different ILRs in GILA exhibit different strengths and weaknesses, and the power of GILA consists exactly of harnessing the strong points of each ILR. For instance, one of the CBLR's strengths is that its performance during prioritization was close to the expert's prioritization. For this reason, the CBLR priorities were used to drive the deconfliction order of the GILA system in the final system evaluation.

6.4 The 4D Constraint Learner-Reasoner (4DCLR)

The *4D Constraint Learner-Reasoner (4DCLR)* within GILA is responsible for automated learning and application of planning knowledge in the form of safety constraints. Examples of such safety constraints include: "The altitude of a UAV over the course of its trajectory should never exceed a maximum of 60000 feet," and "An aircraft trajectory should never be moved so that it intersects a no-fly zone." Note that the maximum altitude is *not* an absolute physical constraint. Constraints are "hard" in the sense that they can never be violated, but they are context-sensitive, where the "context" is the task mission as exemplified in the ACO. For instance, a recommended minimum altitude of an aircraft may be raised if the problem being solved involves the threat of enemy surface-to-air missiles.

The 4DCLR consists of the following two components: (1) the *Constraint Learner (CL)*, which automatically infers safety constraints from the expert demonstration trace and outputs the constraints for use by the other ILRs in the context of planning, and (2) the *Safety Checker (SC)*, which is responsible for verifying the correctness of solutions/plans in terms of their satisfaction or violation of the safety constraints learned by the CL. The output of the Safety Checker is a degree of violation, which is used by the MRE in designing safe subproblem solutions.

Why is the SC needed if the ILRs already use the safety constraints during planning? The reason is that the ILRs do not interact with one another during planning. Because each ILR may not have the domain knowledge, representational expressiveness or learning and planning capabilities to solve the entire input problem, the ILRs output partial (incomplete) solutions, also called *subproblem solutions*. The MRE subsequently composes these subproblem solutions into one final complete solution using search. This final solution needs to be checked because interactions between the subproblem solutions will not emerge until after they have been composed into a single solution, and these interactions might violate constraints (see below for an example).

The approach adopted in the 4DCLR is strongly related to learning control rules for search/planning. This area has a long history, e.g., see (Minton and Carbonell 1987), and has more recently evolved into the learning of constraints (Huang, Selman, and Kautz 2000) for constraint-satisfaction planning (Kautz and Selman 1999). The Safety Checker, in particular, is related to formal verification, such as model checking (Clarke, Grumberg, and Peled 1999). However, unlike traditional verification, which outputs a binary "success/failure," our GILA Safety Checker outputs a *degree* of constraint violation (failure). This is analogous to what is done in (Chockler and Halpern 2004). The difference is that when calculating "degree" we not only calculate the probabilities over alternative states as Chockler and Halpern do, but we also account for physical distances and constraints.

6.4.1 The Constraint Learner and the Representations It Uses

We assume that the system designer provides constraint templates a priori, and it is the job of the Constraint Learner (CL) to infer the values of parameters within these templates.[3] For example, a template might state that a fighter has a maximum allowable altitude, and the CL would infer what the value of that maximum should be.

Simple constraints are unary or binary functions, i.e., $f : \mathcal{C} \rightarrow \Re$ and $g : \mathcal{C} \times \mathcal{C} \rightarrow \Re$, where $\mathcal{C}$ is the set of all concepts (e.g., airspace types or names) defined in the ontologies, $f \in \mathcal{F}$, $g \in \mathcal{G}$, and $\mathcal{F}$ and $\mathcal{G}$ are sets of properties defined over those concepts in the ontologies (Kuter et al. 2007). For example, the maximum altitude that an airspace of type AEW (Airborne Early Warning Area) should fly could be specified as $maxalt(AEW) = 50000$ (feet). Furthermore, the minimum allowable range (called the "band") between the minimum and maximum altitudes of the AEW could be specified as $minAltBand(AEW) = 3200$. Another example constraint might specify that an AAR (Air-to-Air Refueling Area) mission must have at least one CAP (Combat Air Patrol) within a given maximum distance bound, e.g., $maxdist(AAR, CAP) = 65.7$. Binary constraints such as the latter are called *association constraints*; they are often motivated by mission-related asset protection requiring proximity. For example, observing an airspace mission AAR of a combat aircraft could justifiably have a strong influence on the CL's belief regarding the safety parameters for the CAPs involved in escorting the aircraft, but less influence on the parameter values for a reconnaissance

[3]In the future, the CL will learn the templates as well.

Observations

Instance	1	2	3	4	5	6	7	8
Min Altitude (ft)	36000	24000	30000	22000	42000	30000	25000	33000
Max Altitude (ft)	50000	40000	42000	38000	55000	40000	40000	45000

Posterior Distributions after n Observations

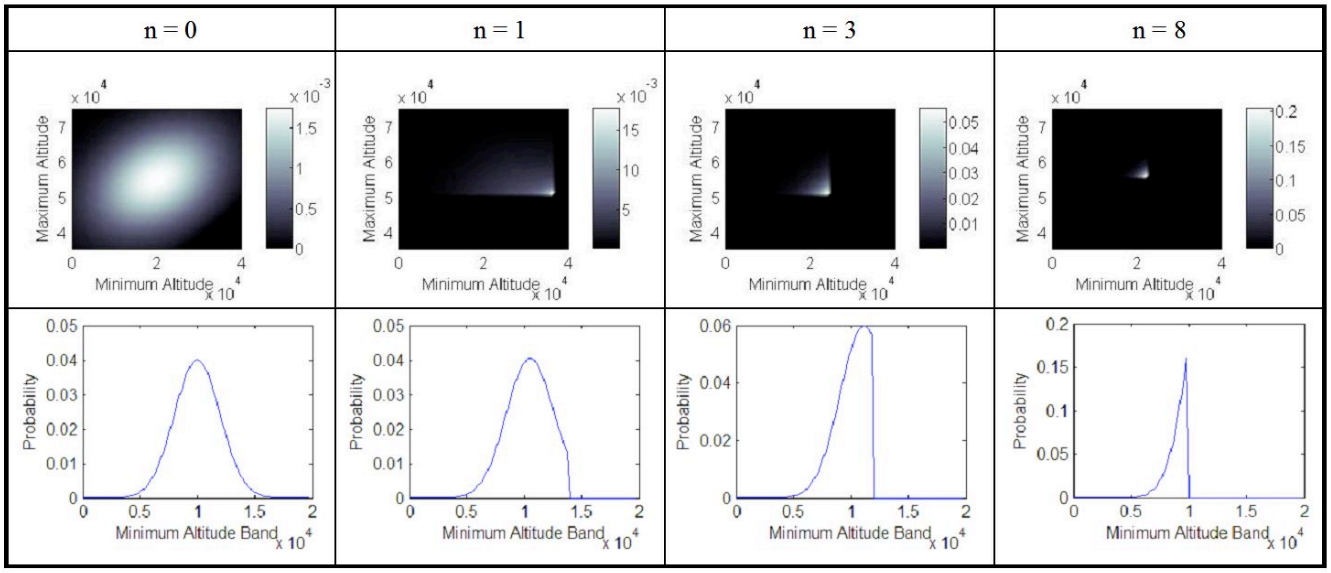


Figure 11: Bayesian constraint learning. The CL observes the airspace instances (top). Bayesian updates lead to tight posterior distributions over the safety constraint values (bottom).

aircraft or unmanned aerial vehicles. Finally, complex constraints are propositional logic expressions involving conjunctions and disjunctions of simple constraints.

Learning in the CL is Bayesian. A discrete probability distribution is used to represent the uncertainty regarding the true value of each parameter. For each parameter, such as the maximum flying altitude for a particular aircraft, the CL begins with a prior probability distribution, $P_f(c)$ or $P_g(c_1, c_2))$, where c, c_1, $c_2 \in \mathcal{C}, f \in \mathcal{F}, g \in \mathcal{G}$. If informed, the prior might be a Gaussian approximation of the real distribution obtained by asking the expert for the average, variance and covariance of the minimum and maximum altitudes. If uninformed, a uniform prior is used.

Learning proceeds based on evidence, e, witnessed by the CL at each step of the demonstration trace. This evidence might be a change in maximum altitude that occurs as the expert positions and repositions an airspace to avoid a conflict. Based on this evidence, the prior is updated using Bayes' Rule to a posterior distribution, $P_f(c|e)$ or $P_g((c_1, c_2)|e)$, and the assumption for the likelihood that the expert always moves an airspace uniformly into a "safe" region. After observing evidence, the CL assigns zero probability to constraint parameters that are inconsistent with the expert's actions, and assigns the highest probability to more constraining sets of parameters that are consistent with the expert's actions. With a modest amount of evidence, this approach leads to tight distributions over the constraint parameters.

An example of Bayesian constraint learning appears in Figure 11. Consider some type of airspace for which there is a minimum altitude, a maximum altitude and a minimum-sized altitude band (i.e., the difference between the minimum and maximum altitudes). From the demonstration trace, the CL observes several expertly placed instances of the airspace type in order to learn the associated constraints. A broad Gaussian prior distribution over constraint values is used for n = 0 observations. After one observation, all constraint settings inconsistent with the observed placement are ruled out, but their posterior distributions are still fairly broad. However, at three and eight observations we witness a tightening of the posterior belief over constraint values due to the properties mentioned in the previous paragraph. Finally, note that Gaussian distributions can be summarized with their mean and standard deviation.

6.4.2 The Safety Checker and Its Outputs for the MRE

The Safety Checker (SC) inputs candidate subproblem solutions from the ILRs, the current ACO on which to try the candidate solutions, and the safety constraints output by the CL; it outputs a violation message. The SC uses its 4D Spatio-Temporal Reasoner to verify whether any constraint is violated by the candidate solution. A violation message is output by the SC that includes the violated constraint, the solution that violated the constraint, specific information about the nature of the violation in the context of the ACO and the expected degree (severity) of the violation, normalized to a value in the range [0, 1].

The expected degree of violation is called the *safety violation penalty*, or simply the *violation penalty*. The SC calculates this penalty by finding a normalized expectation of differences between expert actions (from the demonstration trace) and proposed actions in the candidate solution. The differences are weighted by their probabilities from the posterior distribution learned by the CL. In particular, let e be a value that was obtained from an expert action in the demonstration trace, and let $p_f(c, e) = p_1$ specify that the probability of value e for concept c in the discrete posterior distribution for f is p_1. Let x be the new evidence (i.e., x is the value of the proposed action in the candidate solution). Then the safety violation penalty is calculated as:

$$penalty_{unnormalized} = \sum_e p_f(c, e) \cdot max(0, (e - x)) \tag{1}$$

for a minimum threshold and

$$penalty_{unnormalized} = \sum_e p_f(c, e) \cdot max(0, (x - e)) \tag{2}$$

for a maximum threshold, followed by normalization. The method is identical for relational constraints g. For more complex constraints, the penalty for a conjunction is the average of the penalties of its conjuncts, and for a disjunction it is the maximum of the penalties of the disjuncts. The MRE uses the violation penalty to discard subproblem solutions that are invalid because their penalty is above the *safety threshold*. The safety threshold for violation penalties is specific to a node in the search tree. This tree is used when searching for subproblem solutions, as discussed in the MRE section below.

To illustrate, consider the following example. Suppose a scenario has an airspace of type AEW with unique identifier ACM-K-15, and it has a minimum altitude of 18500 and a maximum altitude of 24500. During planning, the ILRs may need to modify either of these values to remove conflicts with other airspaces. Suppose the SPLR reduces the maximum altitude of ACM-K-15 to 22000 to remove a conflict with another AEW airspace, ACM-K-8. Meanwhile, suppose the CBLR increases the minimum altitude to 20500 to remove a conflict with a UAV airspace. The altitude modifications we see here are considered safe individually. However, when the two plans are merged, in the resulting plan ACM-K-15 has a minimum altitude of 20500 and a maximum of 22000. This results in an airspace corridor with a band of 1500. The question that must be addressed by the SC is whether this is a safety violation.

The SC calculates a violation penalty for the constraint $minAltBand$(ACM-K-15). Continuing our example, suppose that the CL learned a posterior probability distribution that has the following discrete probability points for the minimum altitude difference (i.e., minimum band) of the ACM-K-15:

- $p_{minAltBand}$(ACM-K-15,2000) = 0.100
- $p_{minAltBand}$(ACM-K-15,3200) = 0.350
- $p_{minAltBand}$(ACM-K-15,1200) = 0.070
- ...

Recall that the altitude difference in the modified ACO is 1500. Then the SC calculates the safety violation penalty for this value of 1500 as:

$$\begin{gathered} 0.100 \cdot max(0, 2000 - 1500) \;+\; 0.350 \cdot max(0, 3200 - 1500) \;+ \\ 0.070 \cdot max(0, 1200 - 1500) \;+\; \ldots \;=\; 645. \end{gathered}$$

The value 645 is normalized (using the mechanical range from the ACM-K-15 specifications) to a number between 0 and 1, in this case 0.0775, and is output by the SC as the penalty of the value in the proposed solution for the learned constraint. Note that this methodology is identical for relational safety constraints.

This example illustrates the rationale for verifying the final solution with the SC. In particular, because each ILR creates its plans in isolation, it is unaware of the changes to the scenario that other ILRs are considering in their own solutions. Thus the need arises for this final safety checking pass over the merged solutions. The SC is also invoked earlier – during the merging process. In particular, during plan composition the MRE compares each violation's penalty to the safety threshold, discarding any subproblem solutions that exceed that threshold, thereby pruning and reducing the size of the search space. For example, suppose the safety threshold is 0.05. Then the violation penalty value of 0.0775 in our example above would exceed this safety threshold and the proposed subproblem solution would be rejected.

The 4DCLR learns and verifies many other types of constraints besides those illustrated here. Other classes of constraints include avoidance of no-fly zones, time constraints, geometry constraints that limit the distance that an airspace can be moved in one time step and a variety of mission-driven association constraints. For details, see (Rebguns et al. 2009).

7 Experimental Setup and Results

The GILA system consists of an ensemble of distributed, loosely-coupled components, interacting via a blackboard. Each component is a standalone software module that interacts with the GILA system using a standard set of domain-independent APIs (e.g., interfaces). The distributed nature of the design allows components to operate in parallel, maximizing efficiency and scalability. The GILA system is composed of distributed GILA Nodes, which contain and manage the GILA components. Each node runs in parallel and the components (e.g., ILRs, MRE) are multithreaded within the node. Each node efficiently shares OWL data via the networked blackboard, and the MRE can queue problems for the ILRs, minimizing any idle time. The deployment of components to GILA Nodes is configurable – to optimize performance and scalability. There is no logical limit to the number of nodes or components in the GILA system.

7.1 Test Cases and Evaluation Criteria

The test cases for experiments were developed by Subject Matter Experts (SMEs) from BlueForce, LLC. The experimental results were graded by these SMEs. One expert did the majority of the work, with help from one or two other experts. In the remainder of this section, we use “the expert” to refer to this group. Notice that the expert is independent of the GILA team and is not involved in designing the GILA system.

For the final evaluation, four scenarios, D, E, F and G, were developed. The evaluator randomly chose three of them, namely, E, F and G.[4] In each test scenario, there are 24 Airspace Control Measures (ACMs). There are 14 conflicts among these ACMs as well as existing airspaces. Figures 1 and 2 show testing Scenario E and an expert’s solution, respectively. Each test case consists of three test scenarios for demonstration, practice and performance, respectively.

The core task is to remove conflicts between ACMs and to configure ACMs such that they do not violate constraints on time, altitude or geometry of an airspace. The quality of each step (action) inside the solution is judged according the following factors:

- Whether this action solves a conflict
- Whether this is the requested action
- The simplicity of the action (A conflict may be solved in different ways, and a simple solution should use as few steps as possible and affect the fewest number of conflicts.)
- Proximity to the problem area
- Suitability for operational context
- Originality of the action, which is how creative it is (For example, one action may solve two conflicts. Most of the time, this refers to solutions that the SMEs consider to be quite clever and, perhaps, something that they did not even consider.)

Each factor is graded on a 0-5 scale. The score for each step is the average of all six factors. The final score for a solution is an average of the scores for each step, which is then multiplied by 20 to normalize it in the range [0,100].

GILA’s solution and the expert’s solution for Scenario F are both shown in Table 3. Out of the 14 conflicts to be resolved, four of them are resolved as side-effects of solving the other conflicts in both GILA’s and the expert’s solution. Three of these four conflicts are solved the same way by both GILA and the expert. Among nine other conflicts for which both GILA and the expert provided direct modifications, seven are conflicts for which GILA chose to change the same ACM (in conflict) and make the same type of change as the expert, although the new values are slightly different from the values chosen by the expert. There are two conflicts for which GILA chose to change a different ACM (in conflict), but make the same type of change as the expert. There is one conflict for which GILA changed a different ACM and also made a different type of change. There are four conflicts to which GILA gave the same (or very similar – the difference was 1) priority as the expert. Based on the criteria described above, this GILA solution is scored by the expert with a score of 96 out of 100, as shown in Table 4.

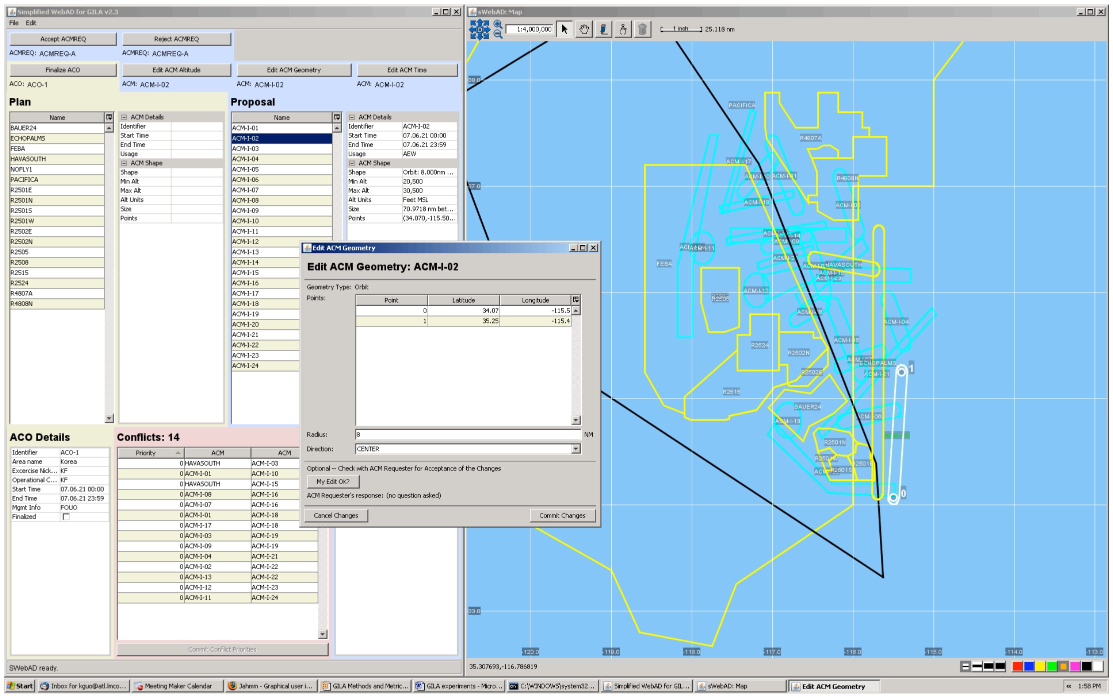
7.2 GILA Versus Human Novice Performance Comparison

Comparative evaluation of the GILA system is difficult because we have not found a similar man-made system that can learn from a few demonstrations to solve complicated problems. Hence we chose the human novices as our base-line. The hypothesis we tested is:

[4]Three other scenarios, A, B and C, were developed and used during debugging throughout system development. They were not officially graded.

Table 3: GILA's and an expert's deconfliction solutions for Scenario F

Expert's priority	Conflict	Expert's Solution	GILA Solution	GILA's priority & Step #
1	ACM-J-18 (SSMS) ACM-J-19 (SOF)	Move 19 from 36.64/-116.34,36.76/-117.64 to 36.64/-116.34,36.68/-117.54	Move 19 from 36.64/-116.34,36.76/-117.64 to 36.6/-116.18, 36.74/-117.48	4
2	ACM-J-17 (SSMS) ACM-J-19 (SOF)	Already resolved	Move 17 from 37.71/-117.24, 36.77/-117.63 to 37.77/-117.24, 36.84/-117.63	1
3	ACM-J-12 (CASHA) ACM-J-23 (UAV)	Change alt of 12 from 16000-35500 to 17500-35500	Change alt of 23 from 1500-17000 to 1500-15125	7
4	ACM-J-11 (CASHA) ACM-J-24 (UAV)	Move 11 from 36.42/-117.87, 36.42/-117.68 to 36.43/-117.77, 36.43/-117.58	Change the alt of 11 from 500-35500 to 17000-35500	10
5	ACM-J-17 (SSMS) ACM-J-18 (SSMS)	Move 17 from 37.71/-117.24, 36.78/-117.63 to 37.71/-117.24, 36.85/36.85/-117.73	Resolved by 17/19 (Step 1)	
6	ACM-J-4 (AAR) ACM-J-21 (SSMS)	Move 4 from 35.58/-115.67, 36.11/-115.65 to 36.65/-115.43, 35.93/-115.57	Move 4 from 35.58/-115.67, 36.11/-115.65 to 35.74/-115.31, 36.27/-115.59	5
7	ACM-J-1 (AEW) ACM-J-18 (SSMS)	Move 1 from 37.34/-116.98, 36.86/-116.58 to 37.39/-116.89, 36.86/-116.58	Move 1 from 37.34/-116.98, 36.86/-116.58 to 37.38/-116.88, 36.9/-116.48	8
8	ACM-J-3 (ABC) ACM-J-16 (RECCE)	Move 16 from 35.96/-116.3,35.24/-115.91 to 35.81/-116.24, 35.24/-115.91	Move 16 from 35.96/-116.3,35.24/-115.91 to 35.95/-116.12, 35.38/-115.65	3
9	ACM-J-3 (ABC) ACM-J-15 (COZ)	Change 3 alt from 20500-24000 to 20500-23000	Change the alt of 15 from 23500-29500 to 25500-29500	9
10	Hava South ACM-J-15 (COZ)	Change 15 time from 1100-2359 to 1200-2359	Change 15 time from 1100-2359 to 1215-2359	2
11	ACM-J-8 (CAP) ACM-J-15 (COZ)	Move 8 from 35.79/-116.73, 35.95/-116.32 to 35.74/-116.72, 35.90/-116.36	Move 8 from 35.79/-116.73, 35.95/-116.32 to 35.71/-116.7, 35.87/-116.3	6
12	ACM-J-3 (ABC) ACM-J-8 (CAP)	Already resolved	Resolved by 8/15 (Step 6)	
13	Hava South ACM-J-4 (AAR)	Already resolved	Resolved by 4/21 (Step 5)	
14	ACM-J-1 (AEW) ACM-J-10 (CAP)	Already resolved	Resolved by 1/18 (Step 8)	



ACMREQ: ACMREQ-A
ACMREQ: ACMREQ-A
ACO: ACO-1
ACM: ACM-I-02
ACM: ACM-I-02
ACM: ACM-I-02

Plan

Name
BAUER24
ECHOPALMS
FEBA
HAVASOUTH
NOFLY1
PACIFICA
R2501E
R2501N
R2501S
R2501W
R2502E
R2502N
R2505
R2508
R2515
R2524
R4807A
R4808N

ACM Details	
Identifier	
Start Time	
End Time	
Usage	
ACM Shape	
Shape	
Min Alt	
Max Alt	
Alt Units	
Size	
Points	

Proposal

Name
ACM-I-01
ACM-I-02
ACM-I-03
ACM-I-04
ACM-I-05
ACM-I-06
ACM-I-07
ACM-I-08
ACM-I-09
ACM-I-10
ACM-I-11
ACM-I-12
ACM-I-13
ACM-I-14
ACM-I-15
ACM-I-16
ACM-I-17
ACM-I-18
ACM-I-19
ACM-I-20
ACM-I-21
ACM-I-22
ACM-I-23
ACM-I-24

ACM Details	
Identifier	ACM-I-02
Start Time	07.06.21 00:00
End Time	07.06.21 23:59
Usage	AEW
ACM Shape	
Shape	Orbit: 8.000nm ...
Min Alt	20,500
Max Alt	30,500
Alt Units	Feet MSL
Size	70.9718 nm bet...
Points	(34.070,-115.50...

ACO Details

Identifier	ACO-1
Area name	Korea
Excercise Nick...	KF
Operational C...	KF
Start Time	07.06.21 00:00
End Time	07.06.21 23:59
Mgmt Info	FOUO
Finalized	☐

Conflicts: 14

Priority	ACM	ACM
0	HAVASOUTH	ACM-I-03
0	ACM-I-01	ACM-I-10
0	HAVASOUTH	ACM-I-15
0	ACM-I-08	ACM-I-16
0	ACM-I-07	ACM-I-16
0	ACM-I-01	ACM-I-18
0	ACM-I-17	ACM-I-18
0	ACM-I-03	ACM-I-19
0	ACM-I-09	ACM-I-19
0	ACM-I-04	ACM-I-21
0	ACM-I-02	ACM-I-22
0	ACM-I-13	ACM-I-22
0	ACM-I-12	ACM-I-23
0	ACM-I-11	ACM-I-24

Edit ACM Geometry

Edit ACM Geometry: ACM-I-02

Geometry Type: Orbit

Points:

Point	Latitude	Longitude
0	34.07	-115.5
1	35.25	-115.4

Radius: 8 NM

Direction: CENTER

Optional -- Check with ACM Requester for Acceptance of the Changes

My Edit OK?

ACM Requester's response: (no question asked)

Cancel Changes   Commit Changes

SWebAD ready.

35.307693,-116.786819

Figure 12: An example of user interface for conducting human test

Table 4: Solution Quality: GILA versus human novices and number of solutions contributed by each ILR

Scenario			Human Novices	GILA	SPLR	CBLR	DTLR
Demo	Practice	Performance					
E	F	G	93.84	95.40	75%	25%	0
E	G	F	91.97	96.00	60%	30%	10%
F	E	G	92.03	95.00	64%	36%	0
F	G	E	91.44	95.80	75%	17%	8%
G	E	F	87.40	95.40	75%	25%	0
G	F	E	86.3	95.00	75%	17%	8%
Average			90.5	95.4	70%	24%	6%

Hypothesis 1 *GILA has achieved 100% human novice performance measured by the trimmed mean score.*

To compare the performance of GILA with novices, we first recruited human volunteers from engineers at Lockheed Martin. After eliminating those who have prior experience with airspace management, we got 33 people for the test. These 33 people were randomly grouped into six groups. Each group was given a demonstration case, a practice case and a test case on which to perform. We used three test cases in six combinations for demonstration, practice and performance. The test could have been more stable if each group could have worked on more than one combination; however, given the availability of the subjects' time, this could not be implemented in our test.

We started with an introduction of the background knowledge. Each of the participants was given a written document that listed all the knowledge that GILA had before learning. They also received GUI training on how to use the graphical interface designed to make human testing fair in comparison with GILA testing. Figure 12 shows a screenshot of the user interface. After the training, each participant was handed a questionnaire to validate that they had gained the basic knowledge to carry out the test. The participants were then shown a video of the expert demonstrations traces on how to deconflict airspaces. Based on their observation, they practiced on the practice case, which only had the beginning and the ending states of the airspaces, without the detailed actions to deconflict them. Finally, the participants were given a performance test case on which they were expected to work. The test ended with an exit questionnaire.

Table 4 shows the scores achieved by GILA and human novices. The score for the human novices shown in the table is the average score of all human novices in a group who are working on the same testing scenario. The score of a solution represents the quality of the solution, which is evaluated by the SME based on the six factors described in Section 7.1. To avoid any experimental bias, the scoring process was blind. The solution was presented in a manner that prevented the expert from determining whether it was generated by GILA or by a human. The maximum possible score for one solution was 100. For example, the first row in Table 4 shows that for experiment EFG (using Scenario E for demonstration, F for practice and G for performance), the average score for human novices is 93.84, while the score of the GILA solution is 95.40. It is shown that based on the average of all six experiments, GILA has achieved 105% of human novices' performance. The trimmed mean score of human novices (which ignores the two highest and two lowest scores) is 91.24. Hypothesis 2 that "GILA has achieved 100% human novice performance (measured by trimmed mean score)" is supported with 99.98% confidence using a t-test.

Here are some general observations of how human novices performed differently from GILA in solving an airspace management problem.

1. GILA sometimes gave uneven solutions, for example 35001 ft instead of 35000 ft. Novices can infer from the expert trace that 35000 is the convention. It seems that the human reasoning process uses a piece of common knowledge that is missing from GILA's knowledge base.
2. Overall, novices lacked the ability to manage more than one piece of information. As the complexity of the conflicts increased, they started to forget factors (e.g., which ACM, which method to change, etc.) that needed to be taken into account. GILA demonstrated a clearly higher level of information management ability in working with multiple conflicts at the same time.

The last three columns of Table 4 show the percentage of contribution made by each ILR in the final solution output by GILA. Note that the 4DCLR is not in this list because it does not propose conflict resolutions, but only checks safety constraint violations. On average, the SPLR clearly dominates the performance by contributing 70% of the final solution, followed by the CBLR which contributes 24%, and finally the DTLR, which contributes 6%. One reason why SPLR's performance is so good is that its rule language, which is based on taxonomic syntax, is very natural and appropriate for capturing the kind of rules that people seem to be using. Second, its lower-level value function captures nuanced differences between different parameter values for the ACM modification operators. Third, it does a more exhaustive search during the performance phase than the other ILRs – to find the best possible ACM modifications. The CBLR does well when its training cases are similar to the test cases, and otherwise does poorly. In the reported test cases, it is found to make poor geometry decisions. The DTLR suffers from its approximate search

Table 5: Comparison of GILA and single ILRs for conflict resolution (Test Scenario: EFG)

	GILA	SPLR	DTLR	CBLR
Conflict Solved	14	14	5	7
Quality Score	95.40	81.2	N/A	N/A

and coarse discretization of the search space. Although it uses the same cost function as the SPLR to search for the solution, its solutions are often suboptimal because it discretizes the parameter space more coarsely than the SPLR. Because of this, it sometimes completely fails to find a solution that passes muster by the 4DCLR, although such solutions do exist in the search space.

7.3 Effect of Collaborations Among Components

To test the importance of the collaboration among various ILRs, we performed two additional sets of experiments. The first set is to run GILA with only one ILR for solving conflicts. The second set is to evaluate the influence of 4DCLR on GILA's performance.

7.3.1 GILA Versus Single ILRs for Solving Conflicts

In this set of experiments, GILA is running with only one ILR for solving conflicts. However, in all these experiments, the DTLR was still used for providing cost information for the MRE, and the 4DCLR was used for safety constraint checking. The hypothesis to test here is:

Hypothesis 2 *No single ILR can perform as well as GILA.*

Table 5 shows that the DTLR is able to solve 5 conflicts out of 14 total conflicts, while the CBLR is able to solve 7 of them. Though the SPLR is able to solve all 14 conflicts, the quality score of its solution (81.2) is significantly lower than the score achieved by GILA as a whole (95.4). The lower score for the SPLR-only solution is caused by some large altitude and time changes, including moving the altitude above 66000. Though there are multiple alternatives to resolving a conflict, usually an action that minimizes change is preferred over those with larger changes. Such large-change actions were not in the solution produced using all ILRs because other ILRs proposed alternative actions, which were preferred and chosen by the MRE. Even when the DTLR's cost function is good, it is unable to solve some conflicts because of its incomplete search. The CBLR fails to solve conflicts if its case library does not contain similar conflicts. The above results support Hypothesis 3 positively. These experiments verify that the collaboration of multiple ILRs is indeed important to solve problems with high-quality solutions.

7.3.2 Performance of the 4DCLR

The performance improvement gained by including the 4DCLR in GILA has been experimentally tested. Here, we summarize the results; for details see (Rebguns et al. 2008). Specifically, we did an experimental investigation of the following hypothesis:

Hypothesis 3 *GILA* with *the 4DCLR generates airspace-deconfliction steps that are more similar to those of the expert than GILA* without *the 4DCLR.*

Two performance metrics were applied in testing this hypothesis. The first, more general, metric used was:[5]

Metric 1: Compare all airspaces moved by GILA and the expert by grouping them as *true positives*, i.e., those moves performed by both GILA and the expert, *false positives*, i.e., those moves that were only done by GILA but not the expert, and *false negatives*, i.e., those that were done by the expert but not by GILA.

The score of GILA, with versus without the 4DCLR, was provided by the following formula:

$$\frac{TP}{TP + FP + FN},$$

[5]See (Rebguns et al. 2008) for the more specific second metric.

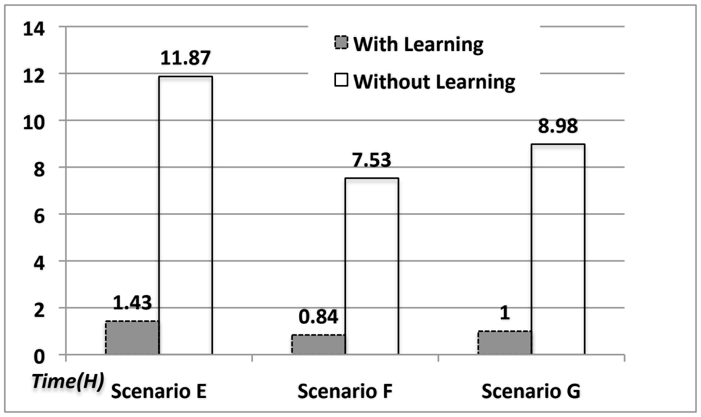
where TP, FP and FN are the number of true positives, false positives and false negatives in an experiment, respectively. The maximum possible score was 1.0, corresponding to complete agreement between GILA and the expert. The lowest score, 0.0, occurred when GILA and the expert chose completely disjoint sets of airspace modifications.

Across five experimental cases, the system generated the following results with the 4DCLR: $TP = 30$, $FP = 18$ and $FN = 22$. Based on this outcome, GILA's score using the first metric was 0.429 when the 4DCLR was included. The score of the system dropped to 0.375 when the 4DCLR was excluded, with the following results: $TP = 27$, $FP = 20$ and $FN = 25$. Clearly, the 4DCLR is helpful for GILA's improved performance.

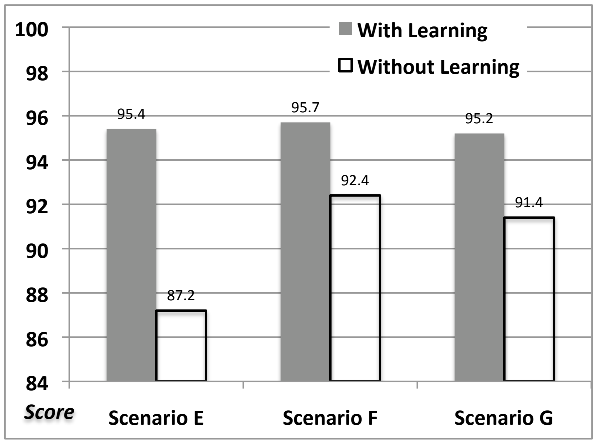
7.4 Effect of Demonstration Content On GILA Performance

Though GILA has achieved quite a good performance score after learning from the expert's demonstration, there is a remaining question of how important the expert's demonstration is. In other words, is GILA solving the problem mainly based on its built-in domain knowledge? To answer this question, we designed the following two sets of experiments.

7.4.1 Performance Without Learning



(a) Performance time



(b) Solution Quality score

Figure 13: Comparison of GILA's performance time and solution score with and without learning

The hypothesis being tested here is:

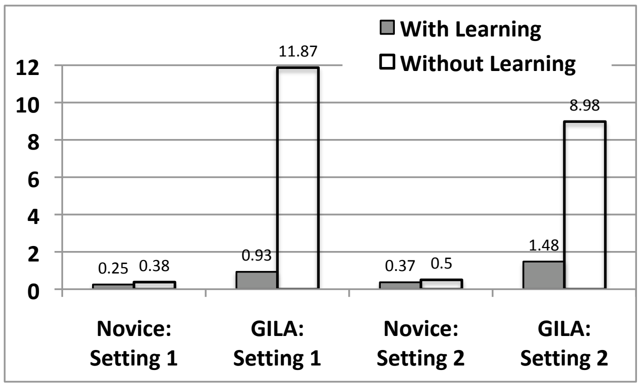
Hypothesis 4 *GILA performs much worse without learning from the expert's demonstration.*

In this set of experiments, we have both GILA and human novices perform on Scenarios E, F and G, without learning from the expert's demonstration. As shown in Figure 13, GILA's performance time increases significantly (about 10 times slower) when it has to solve the same problem without learning. This is due to the fact that GILA has to rely on brute force search to solve the problem. Also, without learning, GILA performs poorly on some of the harder subproblems. The difference in the solution quality score does not seem to be significant; however, small differences in score can mean big differences to the mission. As the SME explains, "An unacceptable altitude in one conflict only brought that conflict resolution's score down to 4.5 [of 5.0]. Put this in the mix for the entire 14 conflicts, and the overall score would change from 4.83 down to 4.81.... yet this could begin a snowball effect that negatively impacted the entire day's ATO." The above analysis of results support Hypothesis 4 positively. Additionally, an analysis of GILA's solutions shows that the improved scores are due to learning. With learning, GILA's solution is nearly identical to the expert's solution.

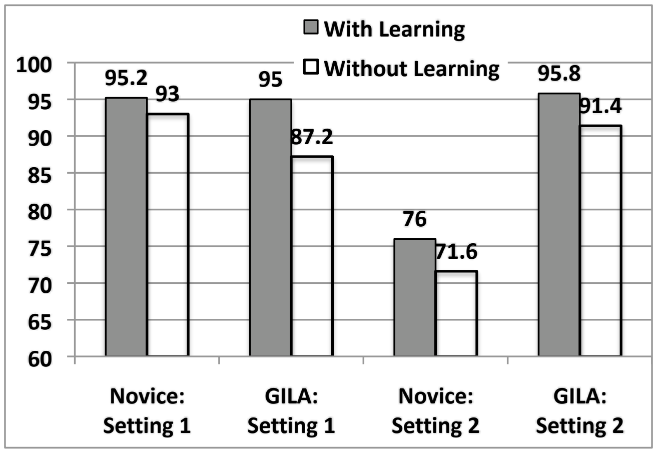
In addition, we have compared novices and GILA, with and without learning on two matched settings:

- Setting 1: perform on Scenario E without learning and on G with learning;
- Setting 2: perform on Scenario G without learning and on E with learning

As illustrated in Figure 14, when perform without learning, novices and GILA show a similar drop in performance in both settings. Without learning, novices take a little bit longer to solve the problem, but not as much as GILA. This is because humans often rely on common sense rather than excessive search to solve problems.



(a) Performance Time



(b) Solution Quality Score

Figure 14: Comparison of the impact of learning on novices and GILA

Table 6: ILRs' Proposed actions after learning from bias trace

	Altitude				Altitude. Geometry				Altitude. Geometry, Time			
PSTEP	Demo Trace	CBLR	SPLR	DTLR	Demo Trace	CBLR	SPLR	DTLR	Demo Trace	CBLR	SPLR	DTLR
SetACMMinAltitude	9	24	35	12	7	6	11	7	4	3	11	5
SetACMMaxAltitude	10	29	35	8	7	7	11	4	6	5	11	4
SetACMPoint	-	-	-	54	10	8	12	-	5	3	14	-
SetStartTime	-	-	-	1	-	-	-	2	2	1	-	2
SetEndTime	-	-	-	1	-	-	-	2	1	-	-	2
SetRadius	-	-	-	14	-	-	-	-	-	-	-	-

7.4.2 Learning from a Biased Expert Trace

To verify that GILA is actually learning what the expert demonstrates, we designed the following bias test. As we described in Section 2, in order to resolve a conflict, there are three types of changes an expert can make to an ACM: altitude change, geometry change and time change. In our bias test, GILA learned from three different expert traces:

1. An expert trace that contains only altitude changes
2. An expert trace that contains only altitude and geometry changes
3. An expert trace that contains all three types of changes

We observed how GILA performed after learning from each of above traces. The hypothesis that was tested is:

Hypothesis 5 *The content of the demonstration has substantial impact on how each ILR solves the problem.*

Table 6 shows how many times each ILR proposed a certain type of change after learning from a specific demonstration trace. Notice that the actions (PSTEPs) for altitude change are *SetACMMinAltitude* and *SetACMMaxAltitude*, actions for geometry change are *SetACMPoint* and *SetRadius*, and actions for time change are *SetSatrtTime* and *SetEndTime*. Both the CBLR and the SPLR learned to strongly prefer the action choices demonstrated by the expert, and they did not generate any choice that they had not been seen in the demonstration. However, biased demonstrations led the DTLR to propose a more diverse set of changes. This was due to the DTLR's unique learning and reasoning mechanism. The DTLR always internally considers all possible changes. If there are only altitude changes in the demonstration, and they do not resolve conflicts during the DTLR's internal search, then it proposes other changes. The above results support Hypothesis 5 positively.

7.5 Effects of Practice

To study the effect of the internal practice phase, we compared GILA's performance score on six scenarios, A, B, C, E, F and G, with and without practice. The average score is 0.9156 with practice, and 0.9138 without practice. The improvement due to practice is small, which shows that GILA has not taken full advantage of practice. In fact, given the time limitation, the following questions have not been addressed thoroughly in the practice phase:

- How should an ILR learn from the pseudo expert trace generated by practice? Currently, the pseudo expert trace is treated as another expert trace, and an ILR learns from it exactly as it learns from the original expert trace. However, this solution does not properly deal with the potentially inconsistent knowledge learned from these two traces.
- How should an ILR share its learned knowledge more actively with other ILRs? Pseudo expert trace actually provides more feedback to ILRs about what they have learned. By analyzing the pseudo expert trace, an ILR can see for each subproblem, whether its proposed solution has been selected. If not selected, the ILR can learn from the reason why it is not selected, and also learn from the actual selected solution.

Though the above questions have not been answered, even now, practice shows good promise for improving solutions. For example, without practice, GILA moves an airspace across the FEBA (Forward Edge of the Battle Area) over into enemy territory, which is not safe. With practice, GILA finds a better way to solve the same conflict – by changing the altitude of one ACM involved in the conflict. Hence we are confident that the practice phase provides a good opportunity for GILA to exercise its learned knowledge and to improve its solution quality.

8 Related Work

GILA is one of two efforts in the DARPA Integrated Learning Program to integrate multiple learning paradigms for learning a complex task from very few examples. The other effort is POIROT (Plan Order Induction by Reasoning from One Trial) (Burstein et al. 2008). POIROT is an integrated framework for combining machine learning mechanisms to learn hierarchical models of web services procedures. Individual learners in POIROT share a common language (LTML - Learnable Task Modeling Language) in which to express their hypotheses (generalizations) and other inferences form the demonstration traces. LTML is based on ideas from OWL and PDDL. Modules can also formulate learning goals for other modules. There is a Meta-Controller that manages the learning goals following the goal-driven learning paradigm (Ram and Leake 1995). The hypotheses generated are merged together into a single hypothesis, using a computational analogy-based method. POIROT incorporates ReSHOP, an HTN planner capable or interpreting planning domains in LTML, generating plans and executing them by invoking web service calls. While GILA's integration approach is "Independent learning with collaborative performance," POIROT's is closer to "Collaborative learning and performance." In fact, GILA's modules (ILRs) are capable of both learning and solving problems; in POIROT, on the other hand, modules only have learning capabilities. Thus, they collaborate during learning, whereas performance is accomplished by a separate module. In machine learning terms, we could see POIROT as using the different modules to explore the same generalization space using different biases, whereas in GILA, each module explores a different generalization space.

FORR (For the Right Reason) (Epstein 1994) is another domain-independent ensemble learning architecture. This architecture assumes initial broad domain knowledge, and gradually specializes it to simulate expertise for individual problem classes. FORR contains multiple heuristic agents called "advisors" that collaborate on problem-solving decisions. A FORR-based program learns both from the performance of an external expert and from practice in its domain. This architecture has been implemented for game playing. The major difference between the FORR architecture and the GILA architecture is that FORR contains one single learner, and all advisors perform based on the same learned knowledge, whereas GILA contains multiple ILRs, and each learns using its own methods and proposes solutions based on its own internal learned knowledge. We believe that multiple diverse learning methods can be advantageous for capturing knowledge from various sources, especially when the expert demonstration examples are very few.

In addition to POIROT and FORR, our work on GILA is related to several areas of research on the integration of learning methods (ensemble learning, multi-strategy learning and multiagent learning) and on learning from demonstration. The rest of this section outlines the connections between GILA and those areas.

Ensemble learning focuses on constructing a set of classifiers and then solving new problems by combining their predictions (Dietterich 2000a). Ensemble learning methods, such as Bagging (Breiman 1996) or Boosting (Freund and Schapire 1996), improve classification accuracy versus having an individual classifier, given that there is diversity in the ensemble. Thus, the focus on ensemble learning is to increase the classification accuracy. Moreover, except for a few exceptions, ensemble learning methods focus on creating multiple classifiers using the same learning method, but providing different training or feature sets. GILA, however, focuses on integrating different learning paradigms in order to reduce the number of training examples required to learn a complex task. Moreover, ensemble learning techniques have been studied for classification and regression tasks, whereas GILA operates on a planning task.

Multistrategy learning studies how to integrate multiple inference types or computational paradigms into a single learning system (Michalski and Tecuci 1994). The motivation for multistrategy learning is that many real-world applications go beyond the capability of "monostrategy" learning methods. In a prototypical example of research in multistrategy learning, Michalski (Michalski 1993) presented a framework of knowledge transmutation operations, with which different learning paradigms could be characterized inside of a common unified framework and achieve integration. GILA, however, avoids that problem by letting each individual module (ILR) learn individually, without merging the learned knowledge.

GILA's ILRs could be considered "agents." Multiagent learning (MAL) studies multiagent systems from a machine learning perspective (Stone and Veloso 2000). Most recent work in MAL focuses on multiagent reinforcement learning. GILA, however, is closely related to work on distributed learning (Davies and Edwards 1995), where groups of agents collaborate to learn and solve a common problem. Work in this area focuses on both the integration of inductive inferences during learning (Davies 2001) (closely related to the POIROT project), and on the integration of solutions during problem solving (Ontañón and Plaza 2007) (which is closely related to the GILA project).

Learning from Demonstration, sometimes called "programming by demonstration" (PBD) or "imitation learning," has been widely studied in robotics (Bakker and Kuniyoshi 1996), and offers an alternative to manual programming. Lau et. al. (Lau 2001) proposed a machine learning approach for PBD based on Version Space algebra. The learning is conducted as a search in a Version Space of hypotheses, consistent with the demonstration example. Human demonstrations have also received some attention to speed up reinforcement learning (Schaal 1996), and as a way of automatically acquiring planning knowledge (Hogg, Muñoz-Avila, and Kuter 2008), among others. Könik and Laird present a *Relational Learning from Observation* technique (Könik and Laird 2006) able to learn how to decompose a goal into subgoals, based on observing annotated expert traces. Könik and Laird's technique uses relational machine learning techniques to learn how to decompose goals, and the output is a collection of rules, thus showing an approach to learning planning knowledge from demonstrations. The main difference between GILA and these learning from demonstration techniques is that GILA analyzes expert demonstrations using multiple learning modules in order to learn as much knowledge as possible, and thus increase its sample efficiency.

This work has been briefly presented in (Zhang et al. 2009). However, this paper includes significantly more details about the components, descriptions of the architectures, discussions of lessons learned and additional experimental results about the effect of demonstration content and the effect of practice.

9 Conclusions and Future Work

In this paper, we presented an ensemble architecture for learning to solve an airspace management problem. Multiple components, each using different learning/reasoning mechanisms and internal knowledge representations, learn independently from the same expert demonstration trace. A meta-reasoning executive component directs a collaborative performance process, during which it posts subproblems and selects partial solutions from the ILRs to explore. During this process, each ILR contributes to the problem-solving process without explicitly transferring its learned knowledge. This ensemble learning and problem-solving approach is efficient, as the experimental results show that GILA matches or exceeds the performance of human novices after learning from the same expert demonstration. The collaboration among various learner-reasoners is essential to success, since no single ILR can achieve the same performance as the GILA system. It has also has been verified that the successful performance of GILA is primarily due to learning from an expert's demonstration rather than from knowledge engineered within the system, distinguishing GILA from a hand-engineered expert system.

The ensemble learning and problem-solving architecture developed in this work opens a new path for learning to solve complex problems from very few examples. Though this approach is tested within the domain of airspace management, it is primarily domain-independent. The collaborative performance process directed by the MRE is domain-independent, with the exception of the approach used to decompose the problem into subproblems. The learning and reasoning mechanisms inside each ILR are generally domain-independent, i.e., they can be transferred to other problem domains. Each ILR can be transferred to other domains as described below.

- The SPLR is specifically designed to be domain neutral. The policy language bias is "automatically" generated from any input domain, thus, transporting the SPLR to other domains would be a straightforward process with minimal human intervention.
- Whereas the CBLR case structure is domain-specific, the learning and reasoning components are domain-independent. Transferring CBLR to another domain would require the identification of the most important features that would be used to represent a case in the new domain along with the representation of the set of steps used to solve a problem in the new domain. A similarity metric and adaptation rules that would operate on these features in the new domain would also be needed. The hierarchical relationships among case libraries would need to match the structure of the new domain.
- The learning algorithm of DTLR is similarly domain-independent while the features are domain-specific. To transfer to a different domain, the following three things need to be redefined: a joint-feature function that gives the features defined on both input $\mathbf{x}$ and output $\mathbf{y}$ to successfully exploit the correlations between inputs and outputs; a loss function that gives a discrepancy

score between two outputs $\mathbf{y}$ and $\mathbf{y}$' for a given input $\mathbf{x}$; and an argmax solver, which is an oracle that gives the best output $\hat{y}$ for a given input $\mathbf{x}$ according to the cost function.

- In terms of task generality, the 4DCLR is easily applicable to any physical-world application that involves physical constraints. Only the ontology and specific domain knowledge would need to be replaced; the algorithms would remain the same. Generalization of the 4DCLR to abstract (non-physical) tasks is a topic for future investigation.

The GILA system can be extended in several ways. For example, GILA could be extended to eliminate the assumption that the expert's demonstration is perfect and that there is no disagreement among experts. Several experts may disagree on similar situations. Each ILR could be enhanced to handle this new challenge. For example, SPLR learning allows negative coverage. For priority and choice learning, the SPLR would choose to learn from the actions of the majority of experts. For margin learning, the SPLR would learn the average margins among experts. The CBLR can currently learn cases from multiple experts who agree or disagree. At performance time, a set of cases that are most similar to the current problem being solved are retrieved, and this set may contain two or more cases with radically different solutions. The CBLR will apply these solutions one at a time, and submit the solution that results in the highest quality airspace deconfliction (i.e., the lowest number of conflicts in the airspace). In the case of an imperfect expert (resulting in a learned case with an incorrect solution) the most similar case will be adapted and applied, and the resulting solution tested. In future work, in order to improve CBLR performance, a case that results in an incorrect solution would be identified, and another case would be adapted and applied in its place. DTLR can be improved by learning search control heuristics and an informed search algorithm that helps find higher quality solutions to its subproblems. We would also incorporate an introspective module that will reason about the quality of the cases learned, based on the solutions that they produce over time, and either remove lower quality cases or flag them so that they are only applied when higher quality cases are not successful. The 4DCLR does not currently consider the situation of multiple experts who disagree, thereby resulting in inconsistent expert traces. In the future, we would like to extend the 4DCLR to address this, by weighting each expert's inputs based on his/her assessed level of expertise.

Another future direction is to introduce further collaboration among the different ILRs. How can each ILR learn from other ILRs more actively? In the work presented in this paper, components are coordinated only by the meta-reasoning module at performance time. As part of our future work, we are exploring the possibility of coordinating the components at learning time by following ideas from goal-driven learning (GDL) (Ram and Leake 1995) (see (Radhakrishnan, Ontañón, and Ram 2009) for preliminary results). We can also provide feedback on each ILR's solution, including an explanation of why its solution was not selected, thereby allowing it to learn from solutions provided by other ILRs. Such collaborations would enhance the performance of the entire GILA system.

References

Aamodt, A., and Plaza, E. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications* 7(1):39–59.

Bakir, G. H.; Hofmann, T.; Schlkopf, B.; Smola, A. J.; Taskar, B.; and Vishwanathan, S. V. N., eds. 2007. *Predicting Structured Data*. Cambridge, MA: MIT Press.

Bakker, P., and Kuniyoshi, Y. 1996. Robot see, robot do: An overview of robot imitation. In *AISB96 Workhsop on Learning in Robots and Animals*. 3–11.

Blum, A., and Mitchell, T. 1998. Combining labeled and unlabeled sata with co-training. In *Annual Conference on Learning Theory(COLT)*.

Breiman, L. 1996. Bagging predictors. *Machine Learning* 24(2):123–140.

Burstein, M. H.; Laddaga, R.; McDonald, D.; Cox, M. T.; Benyo, B.; Robertson, P.; Hussain, T. S.; Brinn, M.; and McDermott, D. V. 2008. Poirot - integrated learning of web service procedures. In *AAAI*, 1274–1279.

Cendrowska, J. 1987. Prism: an algorithm for inducing modular rules. *International Journal for Man-Machine Studies* 27(4):349–370.

Chockler, H., and Halpern, J. Y. 2004. Responsibility and blame: A structural-model approach. *Journal of Artificial Intelligence Research (JAIR)* 22:93–115.

Clarke, E. M.; Grumberg, O.; and Peled, D. 1999. *Model Checking*. MIT Press.

Collins, M. 2002. Ranking algorithms for named entity extraction: Boosting and the voted perceptron. In *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, 489–496.

Daumé III, H., and Marcu, D. 2005. Learning as search optimization: approximate large margin methods for structured prediction. In *Proceedings of the 22nd International Conference on Machine Learning(ICML)*.

Daumé III, H.; Langford, J.; and Marcu, D. 2009. Search-based structured prediction. *Machine Learning Journal* 75(3):297–325.

Davies, W., and Edwards, P. 1995. Distributed learning: An agent-based approach to data-mining. In Gordon, D., ed., *Working Notes of the ICML '95 Workshop on Agents that Learn from Other Agents*.

Davies, W. H. E. 2001. *The Communication of Inductive Inference*. Ph.D. Dissertation, University of Aberdeen.

Dietterich, T. 2000a. Ensemble methods in machine learning. In Kittler, J., and Roli, F., eds., *First International Workshop on Multiple Classifier Systems*, Lecture Notes in Computer Science, 1 – 15. Springer Verlag.

Dietterich, T. G. 2000b. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning* 40(2):139–157.

Doppa, J. R., and Tadepalli, P. 2010. Semi-supervised search-based structured prediction.

Epstein, S. L. 1994. For the right reasons: The forr architecture for learning in a skill domainstar, open. *Cognitive Science Volume 18, Issue 3, July-September 1994, Pages 479-511* 18(3):479–511.

Erman, L. D.; Hayes-Roth, F.; Lesser, V. R.; and Reddy, D. R. 1980. The hearsay-II speech understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys* 12(2):213–253.

Freund, Y., and Schapire, R. E. 1996. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, 148–156. Morgan Kaufmann.

Friedman, J.; Hastie, T.; and Tibshirani, R. 1998. Additive logistic regression: a statistical view of boosting. *Annals of Statistics* 28:2000.

Galvani, B.; Gerevini, A. E.; Saetti, A.; and Vallati, M. 2009. A planner based on an automatically configurable portfolio of domain-independent planners with macro-actions:PbP. In *International Conference on Planning and Scheduling(ICAPS)*.

Goldman, S., and Zhou, Y. 2004. Enhancing supervised learning with unlabeled data. In *Proceedings of the annual IEEE International Conference on Tools with Artificial Intelligence(ICTAI)*.

Hogg, C. M.; Muñoz-Avila, H.; and Kuter, U. 2008. Htn-maker: Learning htns with minimal additional knowledge engineering required. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence(AAAI)*, 950–956.

Huang, Y.; Selman, B.; and Kautz, H. 2000. Learning declarative control rules for constraint-based planning. In *Proceedings 17th International Conference on Machine Learning(ICML)*, 337–344.

Kautz, H., and Selman, B. 1999. Unifying SAT-based and graph-based planning. In *Proceedings of the International Joint Conference on Artificial Intelligence(IJCAI)*, 318–325.

Khardon, R. 1999. Learning action strategies for planning domains. *Artificial Intelligence Journal* 113(1-2):125–148.

Könik, T., and Laird, J. E. 2006. Learning goal hierarchies from structured observations and expert annotations. *Machine Learning* 64(1-3):263–287.

Kuter, U.; Levine, G.; Green, D.; Rebguns, A.; Spears, D.; and DeJong, G. 2007. Learning constraints via demonstration for safe planning. In *AAAI Workshop on Acquiring Planning Knowledge via Demonstration*.

Lau, T. 2001. *Programming by Demonstration: a Machine Learning Approach*. Ph.D. Dissertation, University of Washington.

Martin, M., and Geffner, H. 2000. Learning generalized policies in planning domains using concept languages. In *Proceedings of Seventh International Conference on Principles of Knowledge Representation and Reasoning*.

McAllester, D., and Givan, R. 1993. Taxonomic syntax for first order inference. *JACM* 40(2):246–283.

McDonald, R. T.; Crammer, K.; and Pereira, F. C. N. 2005. Online large-margin training of dependency parsers. In *ACL '05: Proceedings of the 43th Annual Meeting on Association for Computational Linguistics*, 91–98.

Michalski, R. S., and Tecuci, G. 1994. *Machine Learning: A Multistrategy Approach (Volume IV)*. Morgan Kaufmann.

Michalski, R. S. 1993. Inferential theory of learning as a conceptual basis for multistrategy learning. *Machine Learning* 11:111–151.

Minton, S., and Carbonell, J. 1987. Strategies for learning search control rules: An explanation-based approach. In *Proceedings of the International Joint Conference on Artificial Intelligence(IJCAI)*, 228–235.

Muñoz-Avila, H., and Cox, M. 2007. Case-based plan adaptation: An analysis and review. *IEEE Intelligent Systems*.

Ontañón, S., and Plaza, E. 2007. Learning and joint deliberation through argumentation in multiagent systems. In *Proceedings of the 2007 International Conference on Autonomous Agents and Multiagent Systems*, 971–978.

Parker, C.; Fern, A.; and Tadepalli, P. 2006. Gradient boosting for sequence alignment. In *Proceedings of the 21st National Conferenceo on Artificial Intelligence(AAAI)*. AAAI Press.

Radhakrishnan, J.; Ontañón, S.; and Ram, A. 2009. Goal-driven learning in the gila integrated intelligence architecture. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1205–1210.

Ram, A., and Leake, D. 1995. *Goal-Driven Learning*. The MIT Press.

Rebguns, A.; Green, D.; Levine, G.; Kuter, U.; and Spears, D. 2008. Inferring and applying safety constraints to guide an ensemble of planners for airspace deconfliction. In *Proceedings of CP/ICAPS COPLAS'08 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems*.

Rebguns, A.; Green, D.; Spears, D.; Levine, G.; and Kuter, U. 2009. *Learning and verification of safety parameters for airspace deconfliction*. University of Maryland Technical Report CS-TR-4949/UMIACS-TR-2009-17.

Rivest, R. L. 1987. Learning decision lists. *Machine Learning* 2(3):229–246.

Schaal, S. 1996. Learning from demonstration. In *Advances in neural information processing systems (NIPS)*, 1040–1046.

Stone, P., and Veloso, M. M. 2000. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots* 8(3):345–383.

Taskar, B.; Guestrin, C.; and Koller, D. 2004. Max margin markov networks. In *Advances in neural information processing systems(NIPS)*.

Tsochantaridis, I.; Joachims, T.; Hofmann, T.; and Altun, Y. 2005. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research(JMLR)* 6:1453–1484.

Wang, W., and Zhou, Z. 2007. Analyzing co-training style algorithms. In *Proceedings of European Conference on Machine Learning(ECML)*.

Weiss, G., ed. 2000. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press.

Xu, Y.; Fern, A.; and Yoon, S. 2007. Discriminative learning of beam-search heuristics for planning. In *Proceedings of the International Joint Conference on Artificial Intelligence(IJCAI)*.

Yoon, S., and Kambhampati, S. 2007. Hierarchical strategy learning with hybrid representations. In *Proceedings of AAAI 2007 Workshop on Acquiring Planning Knowledge via Demonstration*.

Yoon, S.; Fern, A.; and Givan, R. 2002. Inductive policy selection for first-order MDPs. In *Proceedings of Eighteenth Conference in Uncertainty in Artificial Intelligence(UAI)*.

Zhang, X.; Yoon, S.; DiBona, P.; Appling, D.; Ding, L.; Doppa, J.; Green, D.; Guo, J.; Kuter, U.; Levine, G.; MacTavish, R.; McFarlane, D.; Michaelis, J.; Mostafa, H.; Ontañón, S.; Parker, C.; Radhakrishnan, J.; Rebguns, A.; Shrestha, B.; Song, Z.; Trewhitt, E.; Zafar, H.; Zhang, C.; Corkill, D.; DeJong, G.; Dietterich, T.; Kambhampati, S.; Lesser, V.; and et al. 2009. An Ensemble Learning and Problem-Solving Architecture for Airspace Management. In *Proceedings of Twenty-First Annual Conference on Innovative Applications of Artificial Intelligence (IAAI-09)*, 203–210.